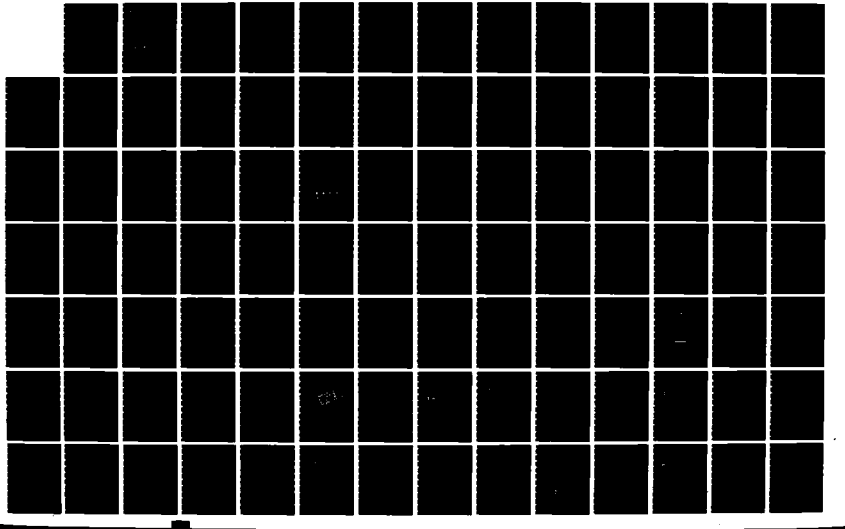
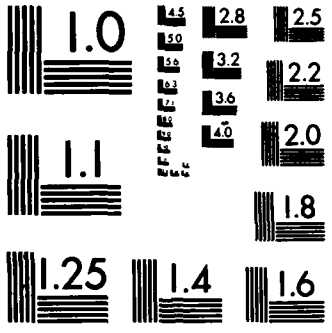
AD-A136 795 AUTOMATED EN ROUTE AIR TRAFFIC CONTROL ALGORITHMIC SPECIFICATIONS VOLUME 3.. (U) FEDERAL AVIATION ADMINISTRATION WASHINGTON DC SYSTEMS ENGINEE.. 1/2

UNCLASSIFIED W P NIEDRINGHAUS ET AL. SEP 83 F/G 17/7 NL





(12)

U.S. Department of Transportation
Federal Aviation Administration
Office of Systems Engineering Management
Washington, D.C. 20591

Automated En Route Air Traffic Control Algorithmic Specifications

FLIGHT PLAN CONFLICT PROBE

Volume 3

September 1983
Report No. DOT/FAA/ES-83/6

Technical Report Documentation Page

1. Report No.	2. Government Accession No.	3. Recipient's Catalog No.
DOT/FAA/ES-83/6	A136795	

4. Title and Subtitle

Automated En Route Air Traffic Control
Algorithmic Specifications
FLIGHT PLAN CONFLICT PROBE Volume 3

5. Report Date
September 1983

6. Performing Organization Code
AES-320

7. Author(s)
W.P. Niedringhaus, I. Frolow, J.C. Corbin, A.H. Gisch, N.J. Taber, F.H. Leiber

8. Performing Organization Report No.
FAA-ES-83-6

9. Performing Organization Name and Address
Systems Engineering Service
Department of Transpotation
Federal Aviaton Administration
800 Independence Ave., S.W., Washington, D.C. 20591

10. Work Unit No. (TRAIS)

11. Contract or Grant No.

12. Sponsoring Agency Name and Address
Same as #9 above.

13. Type of Report and Period Covered

14. Sponsoring Agency Code
AES

15. Supplementary Notes

16. Abstract

This Algorithmic Specification establishes the design criteria for four advanced automation software functions to be included in the initial software package of the Advanced Automation System (AAS). The need for each function is discussed within the context of existing National Airspace System (NAS). A top-down definition of each function is provided with descriptions on increasingly more detailed levels. The final, most detailed description of each function identifies the data flows and transformations taking place within each function.

This document consists of five volumes.. Volume 3, Flight Plan Conflict Probe, contains a functional design for use of trajectory data to predict violations of, separation criteria between aircraft.

The other four volumes of this specification provide design criteria for the following:

- o Volume 1, Trajectory Estimaiton
- o Volume 2, Airspace Probe
- o Volume 4, Sector Workload Probe
- o Volume 5, Data Specification

17. Key Words	18. Distribution Statement
Automation, Air Traffic Control, Automated Decision Making, En Route Traffic Control, Artificial Intelligence, Advanced Automation System	Document is available to the U.S. public through the National Technical Information Service, Springfield, VA 22161

19. Security Classif. (of this report)	20. Security Classif. (of this page)	21. No. of Pages	22. Price
Unclassified	Unclassified		

EXECUTIVE SUMMARY

This specification establishes design criteria for the Flight Plan Conflict Probe (FPCP), a part of the initial automation for the Advanced Automation System of the FAA's next generation air traffic control system. The algorithm provides data for a display to air traffic controllers whenever any two aircraft are predicted to approach each other within certain separation criteria in the horizontal and vertical dimensions. Such a pair of aircraft is called a conflict.

Trajectory Estimation, another function of the Advanced Automation System, models the predicted position of each aircraft as a trajectory, consisting of points in (x,y,z,t) space and the line segments connecting them. Trajectories reflect both pilot intent (his approved flight plan) and current position (radar reports). FPCP automatically tests all trajectory pairs for conflicts.

FPCP is designed to be compatible with current air traffic control procedures. It displays information early enough for controllers to resolve conflicts in a deliberate fashion. It alerts the controller when prompt action is deemed necessary to resolve a conflict.

FPCP determines conflicts by using several separate processes. First, a grid is established to partition the planning region into cells defined in the horizontal and time dimensions. Those cells in the grid through which the trajectory passes are identified and designated as the aircraft's grid chain. The grid chains for all aircraft previously processed by the algorithm are maintained in one data structure. Second, a preliminary or coarse filter compares the grid chain of a specific aircraft to the grid chains of all other aircraft; the aircraft pairs which do not have common cells in their grid chains, and hence are separated by large horizontal distances, are eliminated from further consideration. Third, the remaining pairs are tested to determine if their altitude ranges overlap within the co-occupied cells. For those that do, a final filter analyzes the appropriate segments of the aircraft trajectories associated with the common grid cells. The segments are first checked to see if they overlap in time and violate vertical separation criteria within the common time interval. Those that do are tested for violation of horizontal separation criteria. Information on those segments which violate all of these criteria is maintained and displayed to the controller at the appropriate time.

Some data determined by Flight Plan Conflict Probe are stored in the data base for access by Sector Workload Probe.

TABLE OF CONTENTS

TABLE OF CONTENTS
(continued)

TABLE OF CONTENTS
(concluded)

LIST OF ILLUSTRATIONS

LIST OF ILLUSTRATIONS
(Concluded)

1. INTRODUCTION

The Federal Aviation Administration (FAA) is currently in the process of developing a new computer system, called the Advanced Automation System (AAS), to help control the nation's air traffic. The AAS will consist of new or enhanced hardware (i.e., Central Processing Units, memories, and terminals) and new software.

The new software will retain most or all of the functions in the existing National Airspace System (NAS) En Route Stage A software. The algorithms will need to be recoded and, in some cases, revised. In addition, the new AAS software will contain several new functions that make greater use of the capabilities of automation for Air Traffic Control (ATC). When fully implemented, these new functions are intended to detect and resolve many routine ATC problems.

The initial implementation of the AAS, described in the AAS Specification [1], will provide the ability to detect some common ATC problems. To meet the requirements of the AAS, several new ATC functions need to be postulated and described. Four of these functions are described in this document: Trajectory Estimation, Flight Plan Conflict Probe, Airspace Probe, and Sector Workload Probe [Volumes 1, 2, 3, and 4]. Together, they represent an initial level of automation and the beginnings of the evolution of the ATC system in accordance with the NAS Plan [2]. The NAS Plan presents an overview of the complete set of changes proposed to NAS in the coming decade.

1.1 Purpose

The purpose of this volume is to identify design criteria for Flight Plan Conflict Probe (FPCP). FPCP is one of the advanced automation functions called for in the AAS Specification. These design criteria specified in this volume are based on the existing National Airspace System (NAS) and the specification of the AAS. The AAS specification describes the Flight Plan Conflict Probe function and proposes some high level requirements for this function.

1.2 Scope

This algorithmic specification presents design criteria for a computational framework of Flight Plan Conflict Probe. The framework is a set of algorithms which collectively describe how it may be possible to detect aircraft that are in danger of violating certain separation standards. It may be viewed as a

candidate for consideration in the final design. However, it is not intended to be the complete final design of FPCP in the AAS.

The framework establishes the requirements for input and output data and provides a description of the flow of control of data as it is transferred from input to output. Some of the principal requirements have been identified in the "Operational and Functional Description of AERA 1.01" [3]. To the extent possible, the data are discussed using existing NAS terminology.

1.3 Organization of This Document

The remainder of Section 1 provides a description of Flight Plan Conflict Probe's role in the larger ATC context and in future enhancements of the ATC system. Both the operational considerations and processing methods of FPCP are summarized. Section 2 defines the terminology used in the specification and discusses the factors which influence the design of the algorithms.

Descriptions of the algorithms are contained in Section 3, Flight Plan Conflict Probe Functional Design, and in Section 4, Detailed Description. The Flight Plan Conflict Probe Function, like the other advanced automation functions, is divided hierarchically into subfunctions, components and elements (underlined words in Sections 1 and 2 are critical to the understanding of this specification and can be found in the Glossary, Appendix D). Section 3 specifies the design, environment, and assumptions of the subfunctions (e.g., the Fine Filter), and outlines their components (e.g., Horizontal Check). Section 4 provides a detailed description of each subfunction's components, including their mission, data requirements, and some processing details, and in some cases includes a discussion of a component's elements (e.g., Maneuver Envelope Horizontal Check).

Appendix A defines the data shared by the various subfunctions of FPCP. (Similarly, Volume 5 of this document contains the global data shared by the functions defined in Volumes 1 through 4). Appendix B provides mathematical derivations of certain formulas used in this specification. Supplementary information on "trees," the data structure used by the Coarse Filter, one of the subfunctions of FPCP, follows in Appendix C. Appendix D, as mentioned above, contains a glossary of those terms that are critical to an understanding of this specification.

A Program Design Language (PDL) which describes high level control logic using structured English is used as needed to describe the algorithms in this specification. A description of this PDL is contained in Appendix E. Finally, Appendix F provides a complete list of references.

1.4 Role of Flight Plan Conflict Probe in the Overall ATC System

This section discusses some features of the current ATC system, describes the role of FPCP in the Advanced Automation System, and discusses changes to FPCP that may be appropriate when enhancements to the AAS are introduced.

1.4.1 System Context

The Continental United States airspace is partitioned among 20 Air Route Traffic Control Centers (ARTCCs) or centers, which control regions bounded horizontally by polygons and stretching vertically from the center floor to 60,000 ft. Each center's airspace is further divided into areas, which are in turn divided into sectors. Areas and sectors are polygonal regions with floors (located at specified altitudes or the ground) and ceilings. The sectors of each area are staffed by a group of air traffic controllers (or controllers) specially trained for that area; the area supervisor is the first line supervisor of an area.

In the current ATC System, pilots determine the desired means to reach their destination consistent with current navigational and ATC practices. This intent is then filed with the ATC System as a flight plan (which may be approved by ATC as filed or modified by ATC). Alternatively, flight plans that are executed daily or on a regularly scheduled basis reside in a data base and are filed automatically unless altered or suspended. A flight plan modification may be initiated at any time before or during the flight by a controller or the pilot and must be approved by the controller and the pilot.

Controllers are responsible for monitoring the flights which pass through their sectors and for helping pilots achieve their objectives. They watch a set of symbols representing the aircraft's radar track position as it moves across a plan view display; the aircraft's identity, altitude, and other information are also displayed. Controllers institute control actions as needed, to perform such functions as separation assurance, honoring pilot requests for new routes, rerouting flights to

avoid special use airspaces or severe weather, or queueing aircraft into major terminal areas.

Separation assurance services provided by the current system are described in the FAA's document "Air Traffic Control" [4]. Separation is provided in one of the three dimensions relative to aircraft movement: vertical, lateral, or longitudinal. Separation in any one of these dimensions is sufficient. Vertical separation uses pilot reports of altitude supplemented by barometric data (Mode C reports). Aircraft in level flight may be assigned to specific flight levels, which are designated altitudes (separated by 2000 feet at high altitudes). Level aircraft occupying different flight levels have vertical separation. The controller may provide vertical separation for an aircraft that is maneuvering vertically by issuing altitude restrictions, which direct the pilot to be at, at or above, or at or below a specified altitude at a given point along its flight path. Lateral separation applies to aircraft flying on different routes whose airway widths or protected airspace do not overlap. When routes do overlap, the controller may provide longitudinal separation to assure that the two aircraft reach the region of overlap at two different times or that the aircraft are separated by a specific distance.

The plans of the FAA for the evolution of Air Traffic Control are discussed in "Advanced Automation System, System Level Specification" [1], and in "National Airspace System Plan (NASP): Facilities, Equipment and Associated Development" [2]. According to the NASP, the "early capabilities [of automated Air Traffic Control] will include flight path conflict probe which predicts future aircraft trajectories and examines [them] for potential violation of separation standards." According to the AAS, Flight Plan Conflict Probe is performed "on request or when an amendment is made to an active flight plan."

1.4.1.1 Flight Plan Conflict Probe and AERA

The advanced automation functions for the ATC System are part of an automated system referred to as AERA ("Automated En Route Air Traffic Control"). AERA is to be implemented in several stages, as outlined in "Evolution of Advanced ATC Automation Functions" [5]. Flight Plan Conflict Probe will be implemented as part of the first stage, known as AERA 1 (which is further sub-divided into AERA 1.01 and AERA 1.02). Operational descriptions of the advanced automation functions of AERA 1.01 are given in "Operational and Functional Description of AERA 1.01" [3].

Several other functions of the AAS are related to or interface with Flight Plan Conflict Probe. The Trajectory Estimation function [Volume 1] models the paths of aircraft through space and time for use by other functions including Flight Plan Conflict Probe. These paths are called trajectories. The Airspace Probe function [Volume 2] provides information to air traffic controllers on predicted aircraft violations of restricted and warning areas, military operations areas, areas with terrain obstructions, and special use airspaces. Airspace Probe and Flight Plan Conflict Probe may share some data. The controller may invoke these two functions in tandem when evaluating a proposed routing of an aircraft. Some outputs of FPCP (and of Airspace Probe) are used by the Sector Workload Probe [Volume 4], which provides information to ATC supervisory personnel on measures related to workload in order to assist them in making decisions on sector staffing and on the amount of airspace currently defining the sectors. The common data used by the specifications are described and their relationships are identified in "Data Specification" [Volume 5].

1.4.1.2 FPCP and Other Functions Concerned With Aircraft Separation

The following paragraphs describe three functions or systems which are quite diverse in purpose, source of input data, and look-ahead time, but which share with FPCP the objective of outputting messages whenever certain traffic-related criteria are met. The subsequent section (Section 1.4.1.3) discusses how FPCP satisfies needs not met by the other three systems.

En Route Sector Loading

En Route Sector Loading (ELOD) is a major Central Flow Control enhancement planned for 1983 implementation. Using Official Airline Guide schedules, flight plans, arrival times, and manually entered data, ELOD is to determine areas of projected traffic saturation in sectors and at selected points throughout the U.S. Airspace. The traffic demand is predicted over a longer time period than the period associated with FPCP. An alert message is generated if the projected traffic demand count for any sector or point in a sector exceeds a threshold. This information is provided to the local flow management personnel in the center who may resolve the heavy traffic situation.

ELOD is likely to reduce the incidence of conflicts that would occur without its presence. However, its intent is to estimate

traffic demand on a continental scale rather than to predict separation violations for individual aircraft. It has no information about an aircraft's current position or speed.

Conflict Alert

The currently-implemented Conflict Alert Function, described in "National Airspace System Configuration Management Document: Automatic Tracking" [6], is designed to observe radar-tracked data and alert the responsible controller when certain separation criteria are predicted to be violated. The time thresholds involved are much shorter than ELOD's and too short for Conflict Alert alone to assure routine aircraft separation. Although Conflict Alert is provided with current tracked positions, speeds, headings, and assigned altitudes, it is limited by a lack of knowledge of aircraft intent in the horizontal plane, and hence is subject to false alerts and missed alerts. This limitation, in a sense, is opposite to that of ELOD, which is dependent on intent, but not on tracked position.

Airborne Collision Avoidance Systems

Airborne Collision Avoidance Systems, such as the Traffic Alert and Collision Avoidance System (TCAS), alert an aircraft's pilot to collision threats as described in "Collision Avoidance Algorithm for Minimum TCAS II" [7]. The controller is not involved with the alert. Like Conflict Alert, TCAS is dependent on current relative tracked positions and velocities of nearby aircraft but not on intent. To an even greater degree than Conflict Alert, a TCAS alert (which appears some 30 seconds prior to predicted closest approach) implies that corrective action is necessary; an ATC operational error has probably occurred if the alert involves controlled aircraft. Neither TCAS nor Conflict Alert is adequate for safe separation of aircraft; both serve primarily as collision avoidance systems.

1.4.1.3 Requirement for a Flight Plan Conflict Probe

A problem that exists with all of the above systems is that none combines knowledge of intent with knowledge of tracked position. The difficulty is that confidence in purely tracked data diminishes rapidly as the projection period increases beyond that of Conflict Alert (two minutes), while confidence in flight plan accuracy, reasonably high on ELOD's coarse scale of hours, diminishes rapidly when finer predictions are attempted. The time scales in the intermediate future, however, are most appropriate as thresholds to alert controllers of

conflicts: close enough in the future that corrective action is required, but far enough in the future to allow the controller time to resolve the conflicts in a routine and deliberate fashion. With the introduction of the AAS, it becomes possible for the first time to combine knowledge of tracked position and intent on this intermediate time scale.

1.4.2 Effect of Future AAS Enhancements on Flight Plan Conflict Probe

The role of FPCP may undergo certain modifications in future enhancements of AERA. These modifications are described in detail in "Operational and Functional Description of the AERA Packages" [8].

1.4.2.1 Conflict Resolution

In AERA 1, the Flight Plan Conflict Probe is a detection service only. The controller may respond to FPCP information by planning, verifying and manually uplinking (by voice or data link) resolution maneuvers for the aircraft. Later, the output of FPCP will feed into an automatic resolution service called Conflict Resolution. Conflict Resolution may itself need to invoke FPCP to test proposed resolutions for possible conflicts with some third aircraft. As automation proceeds, the controller's responsibility in planning and coordinating resolution maneuvers will decrease. If necessary, however, he will be able to revert to manual resolution of conflicts using FPCP (as in AERA 1).

Implementation of Conflict Resolution may imply a change in the criteria used by FPCP to determine conflicts. Additional factors may need to be taken into consideration as FPCP's output is increasingly used by other algorithms rather than by controllers alone. These factors are discussed in Section 2.2.4, Separation Criteria.

1.4.2.2 Long Range Probe

A function called Long Range Probe (LRP) is planned for AERA 1.02. The algorithm and the operational use for this function are still under development. The function may require FPCP support in the form of input data. LRP will help a controller decide whether to accept a proposed flight plan or flight plan amendment (e.g., for an off-airway user-preferred route). It will differ from Flight Plan Conflict Probe in that it will not predict conflicts between specific aircraft pairs. Rather, LRP will attempt to indicate the presence of heavy

traffic areas to the controller to help the route approval decision process.

LRP will complement FPCP by providing independent data to the controller concerning proposed flight plan changes. In effect, it will serve as an intermediate function between FPCP (which has a shorter time-frame but uses intent like LRP) and ELOD (which has a longer time-frame but uses statistical projections like LRP).

1.4.2.3 Improved Input Data

As improvements are made on the quality of input to FPCP (predicted aircraft positions), FPCP may use progressively longer look-ahead times for purposes of planning. Also, the separation thresholds used to determine when to alert the controller may be made smaller as ability improves to distinguish false alerts. Possible input enhancements include the following:

- implementation of Mode S data link (a digital two-way air/ground communication system), which will permit better information on aircraft intent and improved weather data (especially for winds aloft)
- improved vertical tracking
- incorporation into the automation data base of position or intent information that is currently discussed and agreed upon by the pilot and the controller via verbal means only

1.5 Flight Plan Conflict Probe Summary

This section describes FPCP from an operational point of view and gives an overview of its internal functioning.

1.5.1 Operational Description

The Flight Plan Conflict Probe informs controllers if the trajectory of an aircraft violates, within a certain time period, specific separation criteria in the horizontal and vertical dimensions, with respect to the trajectory of another aircraft. These criteria differ from the separation standards currently in use by ATC as described in the FAA's document "Air Traffic Control" [4]. If the separation criteria are predicted to be violated for some pair of aircraft and the violation occurs within the specified time period, a conflict is said to

occur, and the controller is presented with a message describing the situation. The controller receives information early enough to produce a resolution and to act promptly, if necessary, in order to eliminate a conflict.

The message may be in the form of text and/or a graphic display. It includes information such as the two aircraft IDs, routes, altitudes, predicted horizontal and vertical miss distances, and the time interval when the separation criteria are predicted to be violated.

FPCP distinguishes two types of conflicts: advisory and priority. A priority conflict indicates that the controller should begin a resolution determination process at once. The process is assumed deliberate (rather than hasty); the criteria allow for complications but not for procrastination. An advisory conflict does not necessarily require immediate attention. Separation criteria for priority conflicts use tighter thresholds than those used for advisory conflicts (see Section 2.1.11, "Advisory and Priority Terminology").

The controller may receive conflict information by requesting a trial probe as defined in "Operational and Functional Description of AERA 1.01" [3]. A trial probe involves the testing of a proposed flight plan change which the controller enters manually. The motivation for a trial probe may be a pilot request for a route change or the testing of a resolution of one conflict to assure that it does not generate any other conflicts. The aircraft's current trajectory remains in the global data base so that all other FPCP processing continues during the trial probe the same way it would have in its absence.

1.5.2 Processing Overview

Flight Plan Conflict Probe is invoked automatically based on an event associated with one specific aircraft called the subject. This event may be that the subject aircraft first enters the data base or its trajectory is altered or extended. Other aircraft in the data base (whose trajectories have at some previous time been processed by FPCP as subjects) are designated as objects. FPCP compares the subject's trajectory against those of each object.

When FPCP is invoked, the subject aircraft's trajectory is compared with the trajectory of object aircraft to rule out objects that, for each interval in time, are separated from the subject by large horizontal distances. The center's entire

airspace, plus a buffer region, comprise the center's planning region which is overlaid by a grid of cells in (x,y,t) space. Certain cells which the subject and object aircraft trajectory approach or pass through are marked "occupied." (The "occupancy" criteria for the subject are different from those for the objects, as explained in Section 2.1.7.) Objects are eliminated from further consideration if they do not share with the subject the occupancy of at least one cell. The remaining objects are listed; they include all aircraft that will closely approach the subject aircraft in the horizontal and time dimensions.

The list is then edited to rule out pairs well-separated in the vertical dimension. The same vertical criteria are used for both advisory and priority conflicts. For the level portions of the subject aircraft's flight, the vertical criteria are that no other aircraft penetrate within a vertical threshold of its assigned flight level. For those portions of the subject's flight involving vertical maneuvers, Trajectory Estimation provides a set of points that define a lower and an upper vertical bound as functions of time. The vertical criteria are that no aircraft penetrate these bounds. In practice, FPCP uses these bounds to associate a single upper and lower bound with each grid cell. As a result, a small percentage of objects may be declared conflicts when they in fact are slightly outside the vertical bounds (the extra computational burden of eliminating them does not appear to be justified, particularly since the upper and lower bounds can be set to reflect this grid-based approximation).

The final sequence of tests use rigorous, mathematical methods to determine whether any of the remaining objects actually violate the separation criteria with respect to the subject. Two simple checks, one for overlap in time and the other for violations in the vertical dimension, are followed by a more rigorous test in the horizontal dimension. FPCP considers the distance between the subject and each object as a function of time. If this distance is predicted to fall below a preselected threshold distance, the advisory/priority criteria are said to be violated. The time at which this distance falls below the threshold is referred to as the time of violation, which is computed and stored in the data base.

Next, the time to display the conflict message is calculated. The message is displayed far enough in advance of the time of violation to allow a reasonable amount of time for the controller to resolve the conflict. The length of this "reasonable" interval of time is a system parameter. It may include

allowances for complications (such as the failure of initially-proposed resolutions), as well as time to coordinate with pilots and other controllers. The time at which the display appears is simply the time of violation minus this parameter.

2. DEFINITIONS AND DESIGN CONSIDERATIONS

Section 2 defines terms that will be used in the following sections and lists design considerations that impact the choice of an algorithm for FPCP.

2.1 System Design Definitions

Some fundamental terms which are used in this and other AERA specifications have already been defined or discussed in Section 1. This section will define additional terms, many of which are used only in this specification. For easy reference, a glossary of both the general and specific terms is included in Appendix D.

2.1.1 Resynchronization

One type of automatic trajectory alteration is resynchronization, defined as the task of recomputing the estimated trajectory of an aircraft when the trajectory is inconsistent with the aircraft's recent history (as determined by radar track data and controller inputs). An AERA function called Conformance Monitoring determines when resynchronizations are necessary and Trajectory Estimation performs them.

2.1.2 Time Horizon, Delta Horizon and Horizon Update

Flight Plan Conflict Probe considers future trajectory information only up to a certain time bound, called the time horizon. The time horizon is far enough in the future that most trajectories within a planning region are encompassed in their entireties (i.e., only a few extend beyond the time horizon). It is advantageous for Flight Plan Conflict Probe (and for Sector Workload Probe) to process trajectory information as far into the future as possible, although neither function depends on the outputs from such processing beyond a certain time limit.

Should a trajectory contain a portion that extends beyond the time horizon, that portion is not processed immediately by FPCP (or SWP). Periodically, at intervals of delta horizon, the time horizon is updated. Its value is incremented by delta horizon. The event is called a FPCP horizon update, and it causes an invocation of FPCP. A determination is made whether a portion of any aircraft's trajectory is encompassed by the updated, but not the original, time horizon. The set of such aircraft is called the horizon subject set. FPCP treats each aircraft, in turn, as the subject, considering only the interval from the original to the updated time horizon. Any

violations of separation criteria found during these invocations must be between two members of the horizon subject set.

2.1.3 FPCP Trajectory Update

A FPCP trajectory update is defined as any of the following three events:

- A trajectory is added to the center's automation data base for the first time.
- A trajectory already in the center's automation data base is resynchronized by the Trajectory Estimation Function. (Possibly the resynchronization was performed in another center and timing information was passed to this center.)
- A trajectory already in the automation data base is altered due to an action by a controller or by another AERA function.

Hereafter, the terms FPCP trajectory update and FPCP horizon update will be shortened to trajectory update and horizon update, respectively. (Note: SWP uses slightly different definitions for these terms; the SWP terms do not appear in this volume.)

In AERA 1, each trajectory update results in an invocation of Flight Plan Conflict Probe (as well as Airspace Probe and a subfunction of Sector Workload Probe).

2.1.4 Segments, Cusps, and Segment Chains

The flight path of an aircraft is actually a continuous, smooth curve in four dimensions. However, aircraft flight paths are approximated by a series of lines (in space-time) called segments, joined together at their endpoints, or cusps, to form a trajectory or a segment chain (Figure 2-1). A trajectory's segment is denoted in the data base by its component cusps.

2.1.5 Holding Patterns and Maneuver Envelopes

It may at times be necessary to delay an aircraft's en route progress. For example, the terminal area at the aircraft's destination may be saturated when the aircraft is due to arrive there. In that case, a controller may direct the aircraft into a holding pattern or hold at a point along its route, causing

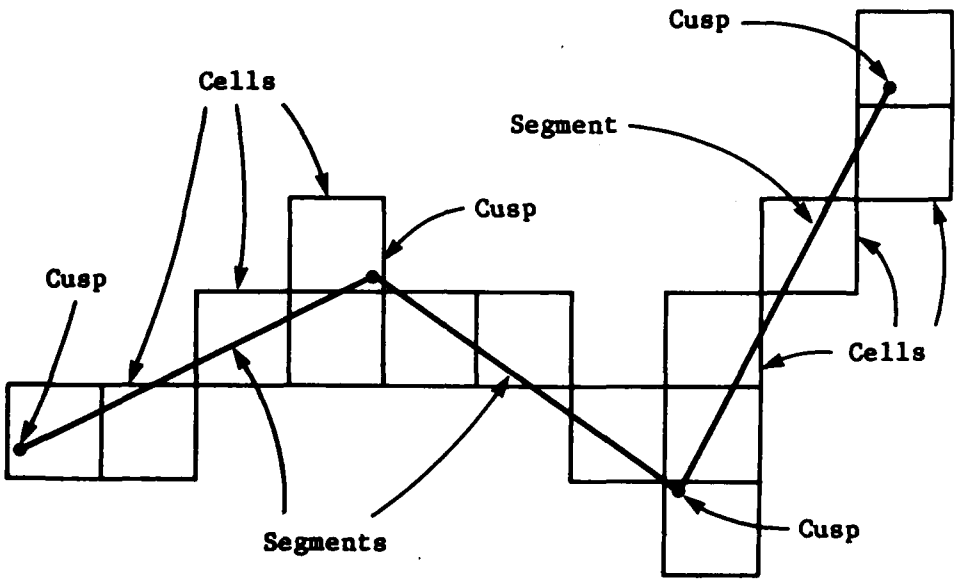


FIGURE 2-1
THE SEGMENT CHAIN AND THE GRID CHAIN FOR A TRAJECTORY

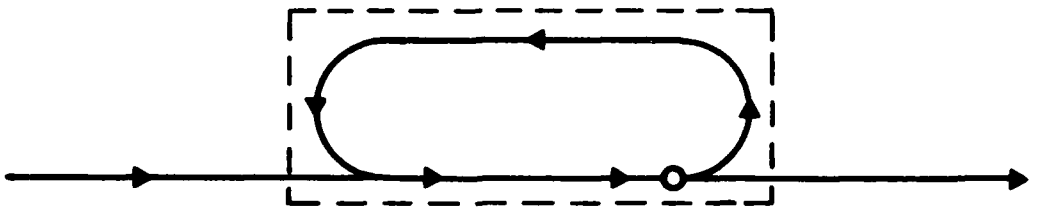
it to maneuver within a specified airspace. The holding pattern may take on any one of two possible general forms:

- a horizontal holding pattern, within which the aircraft maintains level flight
- a holding pattern with vertical extent, within which the aircraft, instructed by the controller, changes altitude using a spiral-like descent (or, conceivably, climb) profile

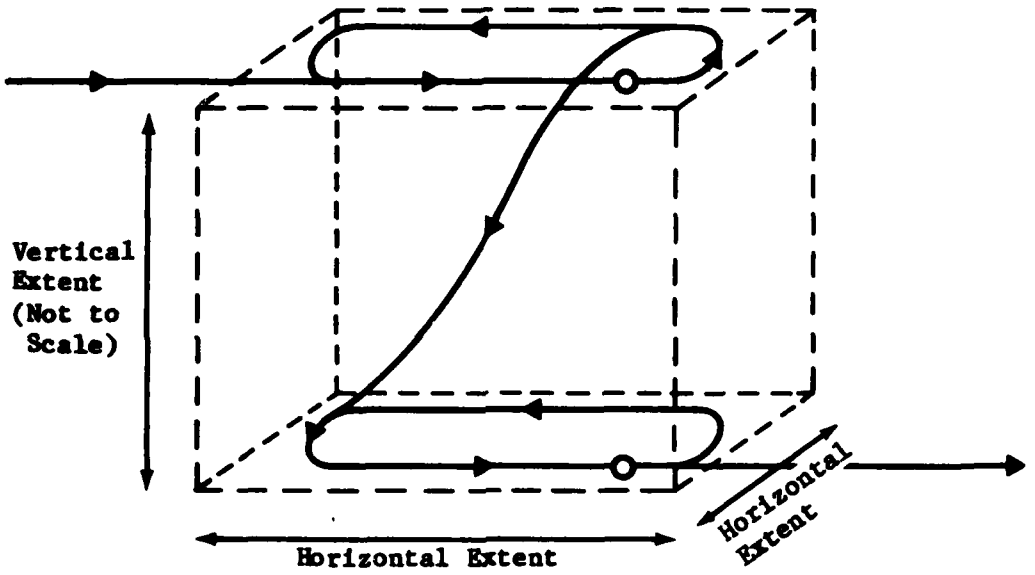
Figure 2-2 illustrates these two types of holding patterns. The figure also shows the geometric structures which enclose these holding patterns and which are constructed by the Trajectory Estimation Function to represent the holds in the data base. The horizontal holding pattern is enclosed by a rectangle and is limited to one flight level, while the holding pattern with vertical extent is contained within a rectangular block in (x,y,z) space which may span many flight levels. Trajectory Estimation provides the vertices of these maneuver envelopes whenever a holding pattern is a part of an aircraft's trajectory. It also provides the spatial and temporal entry and exit points of the holding pattern maneuver. The z coordinates of these points are equal if and only if the hold is horizontal. These two points are generally referred to as cusps, except when they need to be distinguished from regular cusps, in which case they are called holding pattern cusps. The entry and exit points of a holding pattern define a segment of the trajectory. Whenever it is necessary to distinguish such a segment from one defined for a trajectory with no holds, the segment is referred to as a holding pattern segment (versus a regular segment). Otherwise, it is simply referred to as a segment, as described in Section 2.1.4 (Segments, Cusps, and Segment Chains).

2.1.6 Airspace Grid and Its Cells

It is useful to represent the planning region airspace by a grid in (x,y,t) space, called the airspace grid. The discrete compartments in the airspace grid are called the grid cells, or simply cells. Each cell is bounded by surfaces parallel to the x, y, and t axes (the projection of a cell into the (x,y) plane is simply a square). The horizontal and time dimensions of the cells are system parameters. Given these parameters, a cell may be uniquely defined by three numbers corresponding to the positions of the edges of the cell along each of the axes.



(a) Regular Holding Pattern



(b) Holding Pattern with Vertical Extent

Legend:

O Entry and Exit Points (Cusps)

- - - Structure Edge

——— Aircraft Path

FIGURE 2-2
GEOMETRIC STRUCTURES ENCLOSING
TYPICAL HOLDING PATTERNS

It is assumed that a coordinate system is used which allows a rasonably convenient means of interfacing data among centers, even over wide geographical areas. Effects of the curvature of the earth may cause the shape of the cells to deviate slightly from their "ideal" square shape assumed here. Such effects are not significant to the algorithm and are not discussed further in this volume.

2.1.7 Cell Occupancy, Grid Chains, and Buffer Cells

As the trajectory of an aircraft traverses the airspace grid, its segment chain intersects or occupies a sequence of cells. In this section, cell occupancy is carefully defined so that the subject and a given object are considered in conflict only if they "occupy" at least one common cell. An effective strategy in FPCP is to use a minimal number of cells to represent trajectories for object aircraft, which are many in number, while placing the burden of providing separation assurance on the single subject aircraft. This strategy has the advantage of greatly reducing the storage requirements for lists of occupied cells, which are called grid chains. The cells drawn in Figure 2-1 form the grid chain for the segments shown.

The cell list containing a minimal number of cells used in conjunction with object aircraft is called the sparse grid chain. The criterion for determining that a cell is to be considered a member of the sparse grid chain is that the current aircraft segment chain penetrates the cell in such a manner that at least two octants of the cell are intersected (octants play a critical role in the Grid Chain Generator described in Sections 3.3.1 and 4.1). Less sparse representations may be used but the associated FPCP algorithms are made less efficient.

It is possible to construct pairs of trajectories which violate separation criteria while not producing overlapping sparse grid chains. Therefore, sparse grid chains by themselves are not suitable for FPCP separation assurance. Certain additional buffer cells, neighbors of those in the sparse grid chain, are selected for the subject aircraft. The list of these plus the sparse grid chain is called the buffer grid chain. Enough buffer cells are added to assure that for each violation of separation between the subject and an object, the subject's buffer grid chain and the object's sparse grid chain contain at least one cell in common. Finally, we may define what is meant by an occupied cell: for an object aircraft, it is any cell in its sparse grid chain, and for the subject aircraft, any cell in its buffer grid chain.

When the cell size in the x and y dimensions is suitably chosen, the buffer grid chain need include only the cells of the sparse grid chain and all of their orthogonal and diagonal nearest neighbors.

2.1.8 The Sparse Subject Tree, Buffer Subject Tree, and Allobject Tree

Flight Plan Conflict Probe maintains a data structure called a tree to represent the sparse or buffer grid chains of one or more aircraft. In graph terminology, a tree is a set of nodes (called tree nodes) connected by edges with the following properties:

- Each tree node is assigned a nonnegative integer, called the tree node's level, which represents the tree node's position along the vertical axis of the tree.
- Exactly one tree node, called the root, has a level of 0.
- Each edge connects tree nodes whose levels differ by one. The lower level tree node (one closer to the root) is called the parent of the higher level tree node (one further away from the root), which, in turn, is called the child of the lower level tree node.
- Each tree node other than the root has exactly one parent. The root has no parent.

A tree node's ancestors consist of its parent, parent's parent, and so on, all the way to the root. Its descendants are its children, children's children, etc. A tree node with no children is called a leaf.

In relational data base terms, a tree is represented by a table showing all the parent-child relationships (tree edges). Both parent and child are keys. In global tables, the tree nodes are always referred to (in full) as tree nodes, to distinguish them from nodes in ATC terminology, where the word has an alternate meaning. Although the ATC meaning of "node" does not occur in this document, it is well-established in the current ATC system and may be used in future documents describing later versions of AERA. In the text and in local tables in this volume, the shorter form "node" will be used hereafter; the term is always intended to mean a tree node.

Each node is associated with a portion, or block, of the grid. The root of the tree represents the entire grid. Blocks associated with a node's children are called octants or subblocks. The leaves, which all have the same level, represent the cells of the grid. Each node's block contains the blocks of the node's descendants and is contained in the blocks of the node's ancestors. Any set of cells in the grid (and, in particular, the grid chains) may be represented by a tree containing a leaf for each cell plus the ancestors of each such leaf.

Let n be the smallest positive integer such that a square 2^n cells wide (along each of the x and y axes) encloses the entire planning region. Consider this square extended to a cube in (x,y,t) space by projecting it along the time axis by 2^n cells. The cube is a block in the above sense. The block may be divided in half along each axis to form eight octants - "northwest-early," "northwest-late," etc. Each octant may itself be divided in a similar way. A total of n divisions can be carried out before the (indivisible) cell level is reached. Figure 2-3 illustrates the procedure for n = 3, the level of each leaf.

In FPCP, only the blocks containing occupied cells (called occupied blocks) are actually represented by nodes in the tree. Nodes corresponding to blocks whose cells are all unoccupied are not represented. The occupied cells form the leaves of the tree. Figure 2-4 indicates how the tree representation is accomplished in two dimensions. Several journal articles [9, 10, 11, 12] provide some useful theoretical results and algorithms for quad trees (two-dimensional versions of the three-dimensional octal trees used in FPCP).

Depending on the separation parameters, there may be several times as many cells along the t axis as along the x and y axis. That is, the time from now to the time horizon, divided by the width of the cell in the time dimension, may be larger than the number of cells needed to span the planning region horizontally. The discussion is somewhat simplified if the ratio of the former to the latter is assumed to be a small integer power of two. If the entire airspace grid were simply halved repeatedly in each of the x, y, and t dimensions, cell width would be reached in the x and y dimensions before cell width is reached in the t dimension. It is desired, however, to reach cell width in all dimensions at the final division, corresponding to the tree's leaves. Therefore, the first few branching levels from the root are binary, not octal, and represent divisions of the grid on time alone. Octal branching

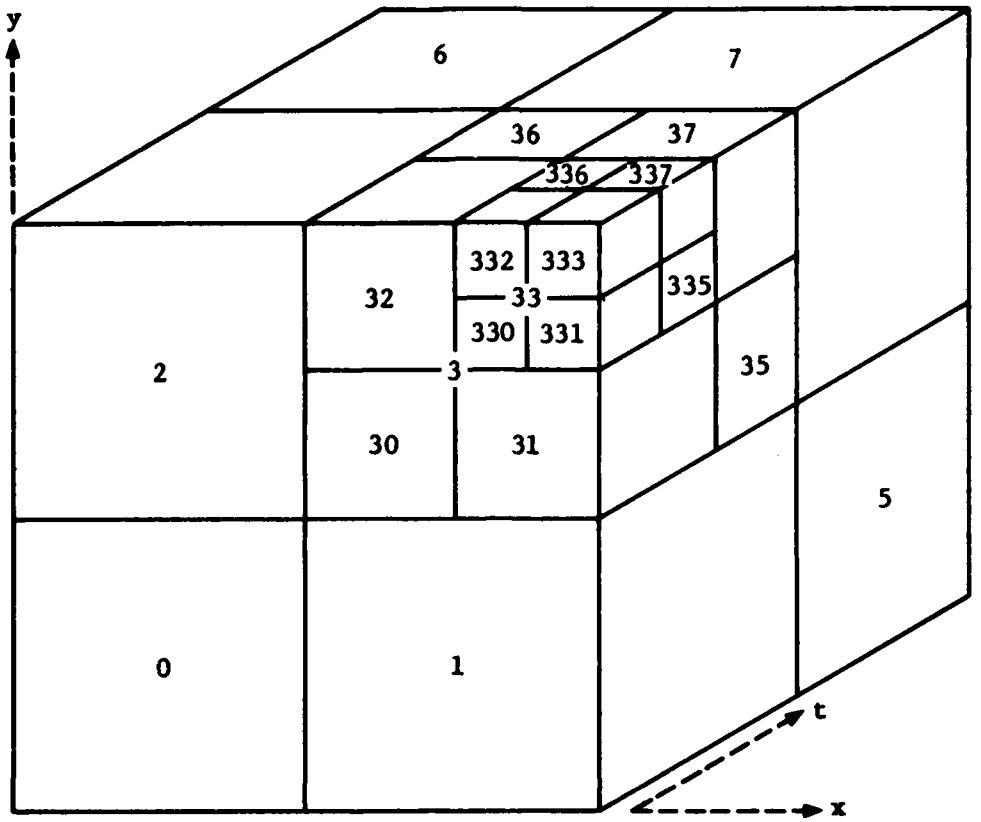


FIGURE 2-3
AIRSPACE GRID IN THREE DIMENSIONS

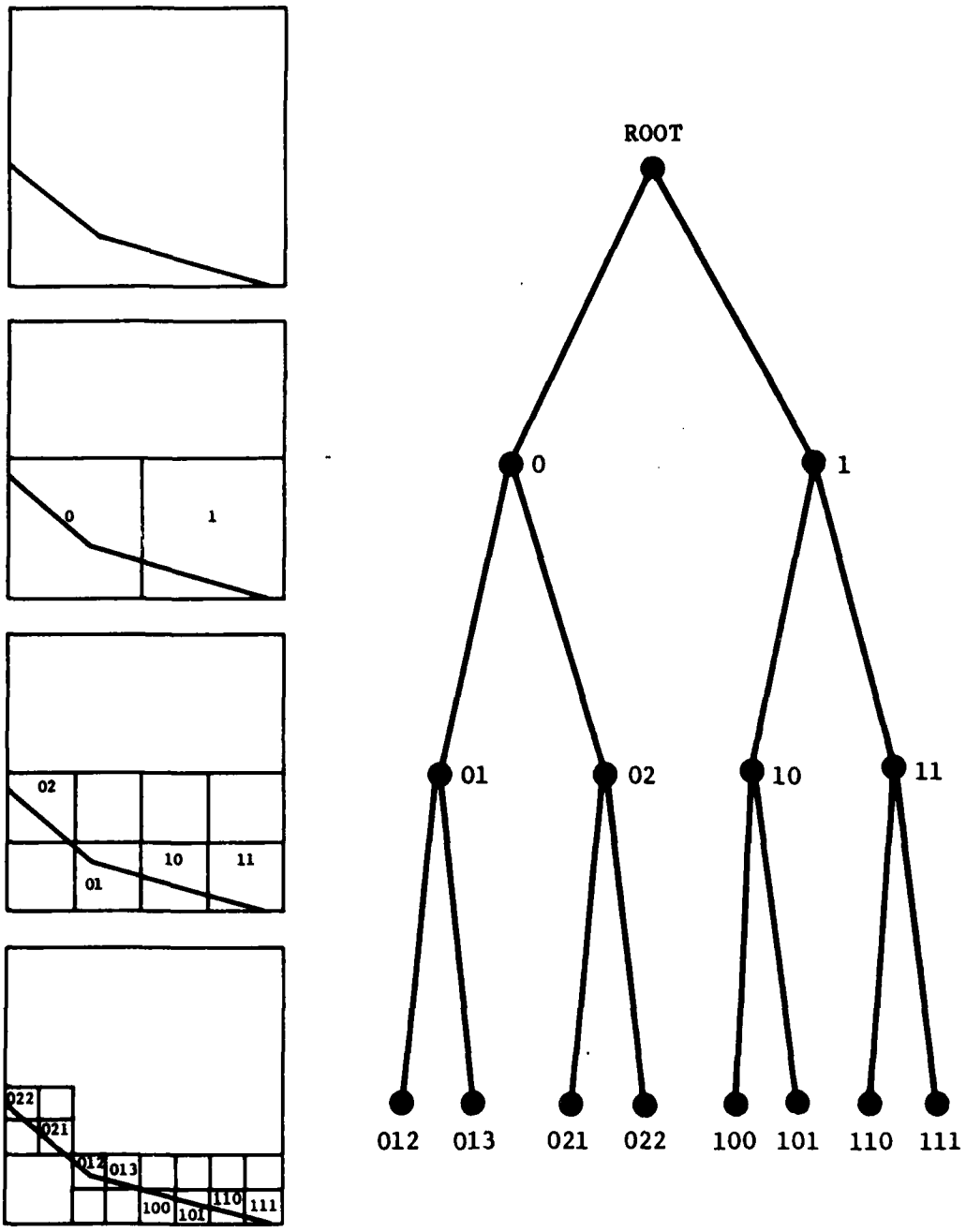


FIGURE 2-4
TREE REPRESENTATION OF THE AIRSPACE GRID IN TWO DIMENSIONS

begins only when blocks (of size 2^n) are reached that have the same number of cells in all three dimensions (Figure 2-5).

When FPCP is invoked, it creates two (local) trees, a buffer subject tree and a sparse subject tree, which represent the cells of the subject aircraft's buffer grid chain and of its sparse grid chain, respectively. FPCP also maintains another tree called the allobject tree, which represents the union of the cells in the sparse grid chains of all aircraft in the planning region.

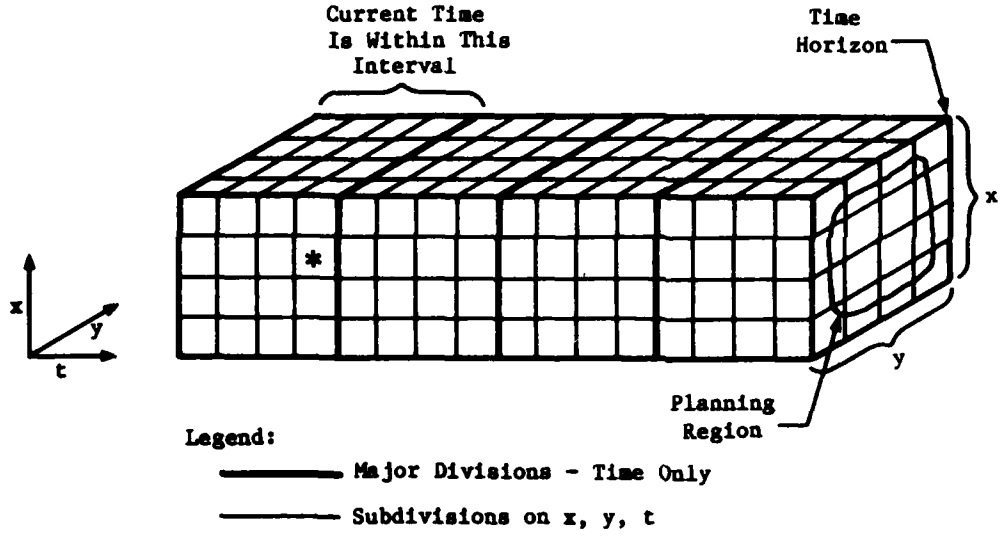
2.1.9 Nominees and the Coarse Filter

A working hypothesis in this formulation of the FPCP is that a detailed comparison of a subject trajectory with each object trajectory in the planning region, using a mathematically rigorous statement of aircraft intent and the FPCP separation criteria, is computationally inefficient. Some prescreening is needed to eliminate subject-object pairs from further consideration when they "obviously" are well separated (in space or time). The existence of prescreening is justified, in principle, since it reduces the overall processing time of the FPCP algorithm. In practice, it has a secondary justification—the data used for prescreening provide useful information for Sector Workload Probe.

A first-level screening, called a Coarse Filter, is performed in order to eliminate object aircraft that are well separated from the subject. The remaining objects, called the nominees or nominee aircraft, are listed for further screening. No objects that violate the FPCP separation criteria are omitted from this list.

2.1.10 Encounters and the Fine Filter

The Coarse Filter produces nominees which may not actually violate the FPCP separation criteria. Another filter, called the Fine Filter, invokes algorithms that, through more rigorous mathematical analyses conducted on subject-nominee segment pairs, can identify those nominees whose trajectories violate both the FPCP horizontal and vertical criteria with that of the subject aircraft. If such a violation is found to exist, the event is called an encounter and the aircraft intruding into the subject aircraft's airspace is called an encounter aircraft. If neither aircraft's trajectory changes, the encounter will, with the passage of time, become an (advisory) conflict.



(X, Y, T) GRID SHOWING GREATER EXTENT IN T DIMENSION

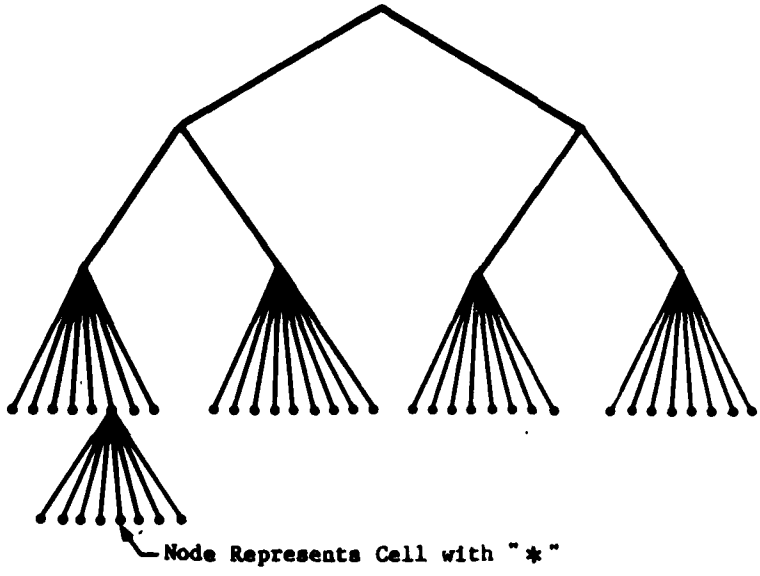


FIGURE 2-5
ILLUSTRATION OF TREE WITH EARLY DIVISION ON T ONLY: LATER DIVISION ON X, Y, T

2.1.11 Advisory and Priority Terminology

The Fine Filter uses two distinct sets of criteria in its testing for conflicts in the horizontal and time dimensions, one for advisory and the other for priority conflicts (in the vertical dimension, the same criteria are used for both types of conflicts). Each set of criteria consists of a threshold for the horizontal test and another for the time to display test. In the horizontal and time dimensions, a pair of aircraft are said to be in conflict if the horizontal separation distance between the aircraft is less than the horizontal separation threshold and the time until the separation between the aircraft reaches this horizontal threshold is less than the time threshold.

The first set of criteria consists of the horizontal separation threshold or advisory Seph (Separation horizontal) and the time threshold or advisory Sept (Separation time).

The advisory time of violation is the time at which separation first falls below the advisory Seph. An advisory message, featuring information on a conflict, is displayed to the controller at the display-as-advisory time, which is the later of

- the time prior to the advisory time of violation by the amount of the advisory Sept
- the current time

If the time of violation is currently less than the current time plus the advisory Sept, an advisory message may be displayed at once and the criteria discussed below are tested. Figure 2-6 illustrates the relationship between Seph and Sept for advisory and priority conflicts. In Figure 2-7, the display-as-advisory time and advisory time of violation are identified relative to the separation between a pair of aircraft and the point of closest approach.

The second set of criteria are analogous to the first, except the values of the thresholds, priority Seph and priority Sept, are smaller. The priority time of violation is the time at which separation falls below the priority Seph. Information on such an encounter would be displayed to the controller in a priority message at the display-as-priority time, which is the later of

- the time prior to the priority time of violation by the amount of the priority Sept

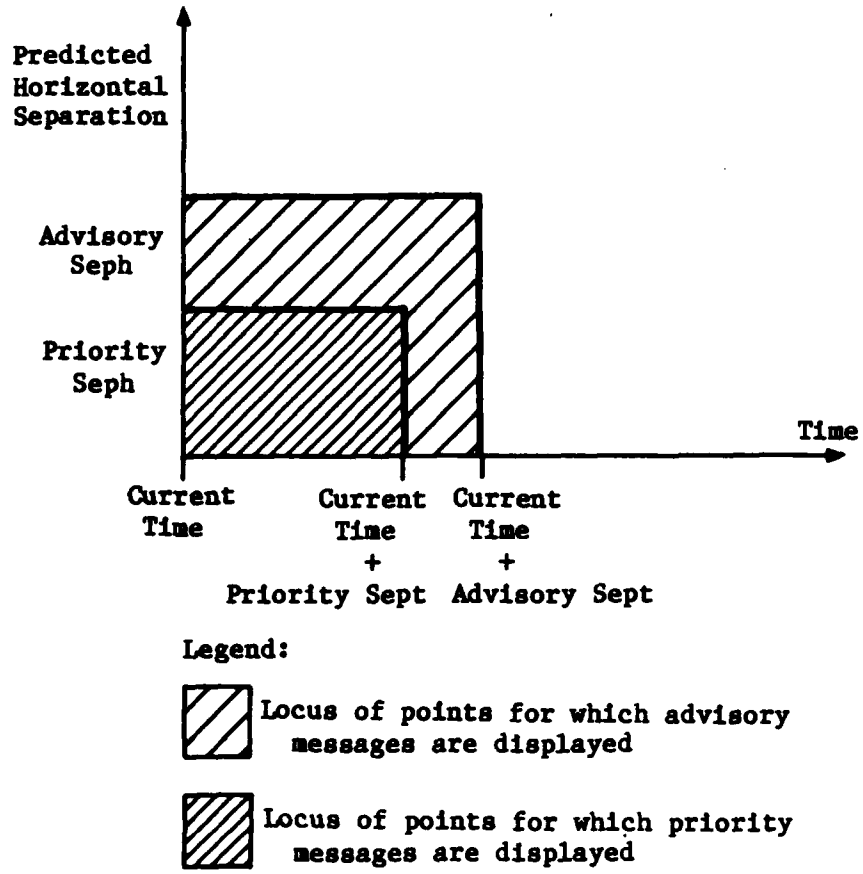


FIGURE 2-6
RELATIONSHIP BETWEEN ADVISORY AND PRIORITY SEPARATION AND DISPLAY CRITERIA

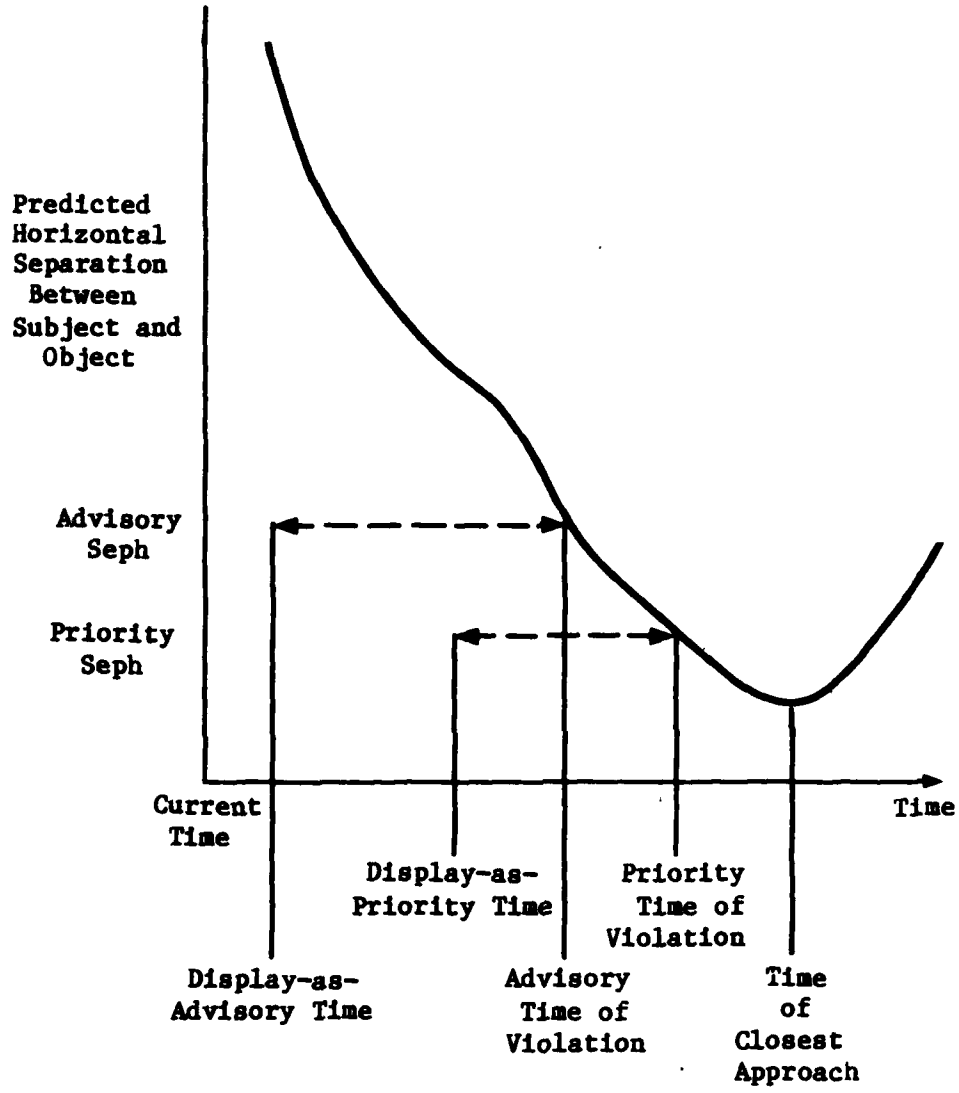


FIGURE 2-7
CRITICAL TIMES ASSOCIATED WITH ADVISORY AND PRIORITY VIOLATIONS

- the current time

2.2 Design Considerations

This section lists considerations that must be taken into account when designing an algorithm for FPCP.

2.2.1 Minimal Required Controller Knowledge of Algorithms

The FPCP algorithm has been designed so that the controller may use the information generated without knowledge of the algorithm's details. An operational understanding of the algorithm may be useful so that the controller can interact and utilize the functions and their outputs.

2.2.2 Display Format

This specification does not address the formats that may be used to display FPCP data outputs or to enter requests for a trial probe or a list of encounters. It has been written under the assumption that the display is flexible and easy to use (perhaps menu driven), has some standard editing capability, and can satisfy both the controller who wishes to explore every feature as well as the controller who wishes to minimize the time required to learn how the function is used.

2.2.3 Considerations Involving Uncertainties in Aircraft Position

FPCP performance is only as good as the automation data base, including approved flight plans, weather, aircraft performance information, and estimated trajectories. Should a controller approve a modification to a flight plan but fail to enter the information into the data base, any displayed conflicts for the aircraft will likely be erroneous since they are based upon an obsolete trajectory. Also, actual conflicts resulting from the new flight plan may not be detected and displayed.

There are ways that erroneous trajectories can be detected automatically for manual or automatic revision. In AERA 1, disparities in the longitudinal (along track) direction between the estimated trajectory and radar return for an aircraft are corrected by resynchronization (although not immediately due to tracker lag). Trajectory Estimation modifies the stored value of the aircraft's speed to account for the observed error, and recomputes the trajectory. Lateral (across track) disparities are not corrected automatically, since when such an error occurs, an aircraft can no longer be assumed to be following

its flight plan. A message indicating the disparity is displayed to the controller, who may update the data base. In the meantime, systems that do not use the automation data base (such as Conflict Alert and TCAS, described in Section 1.4.1.2) serve as backups. In case of conflicts involving an aircraft with a lateral disparity, an indicator of the disparity accompanies the display of conflict information.

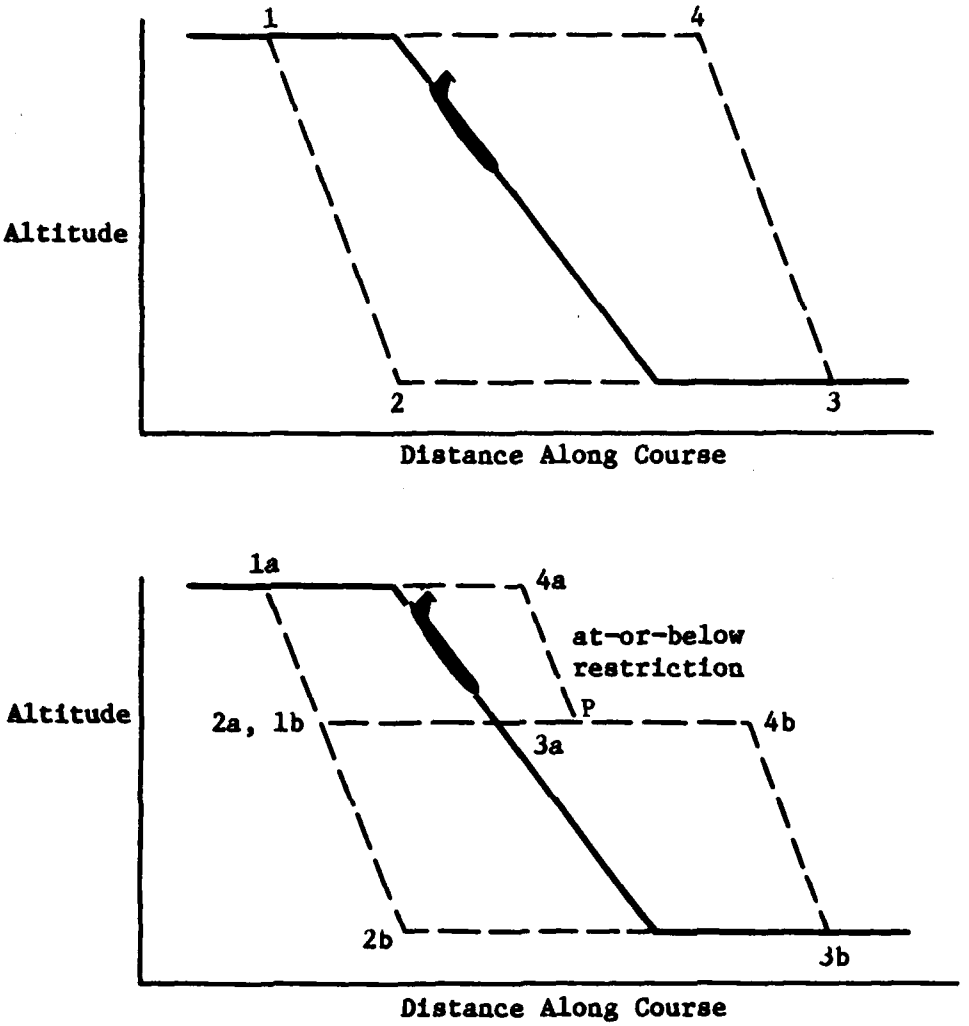
2.2.4 Separation Criteria

The values of FPCP horizontal separation criteria (advisory and priority) are an issue that is still the subject of study. They influence directly the optimal selection of cell size in the horizontal (x and y) dimensions. The AAS Contractor should regard separation parameters and cell size as constants to be determined. A number of factors may influence the choice of the horizontal criteria. The rate of false alerts is one factor. Another is the trade-off between resynchronization rate and the advisory (and priority) Seph: as more error is allowed before triggering a resynchronization, FPCP must use larger thresholds to assure separation just prior to the resynchronization. It is possible that the time elapsed since the last resynchronization is a factor also.

In later stages of automation, factors that are less easily measured may become important in setting horizontal criteria. These factors may include the overall complexity of the situation, additional conflicts caused by possible resolutions, current/ future controller workload, metering plans, etc.

In the vertical dimension, separation criteria for FPCP are more easily defined: all altitudes that an aircraft may legally reach (subject to controller restrictions) and can physically attain (subject to considerations of weather, weight, etc) must be protected.

Trajectory Estimation provides information to FPCP regarding vertical separation criteria. An upper and lower bound are associated with each vertical maneuver. These bounds, like trajectories, consist of points or vertices in four-space connected by lines. The upper portion of Figure 2-8 illustrates a simple case, where a descent has been cleared with no intermediate restrictions. To achieve vertical separation, other aircraft must pass outside the bounds. Trajectory Estimation provides four vertices which identify a vertical maneuver envelope. The four vertices are as follows:



Legend:

1 = left upstream vertex
2 = left downstream vertex
3 = right downstream vertex
4 = right upstream vertex
a = first maneuver envelope
b = second maneuver envelope

FIGURE 2-8
UNRESTRICTED AND RESTRICTED VERTICAL MANEUVER ENVELOPES

1. The earliest point at which the maneuver may begin (left upstream vertex in a descent)

2. The point at which the desired altitude is reached, if the aircraft maneuvers at its maximum vertical rate as early as possible (left downstream vertex)

3. The latest point at which the maneuver may begin (right upstream vertex)

4. The latest point at which the desired altitude must be reached, or some point far in the future if no such constraints exist upon the maneuver (right downstream vertex)

The controller may direct the pilot to achieve a given altitude (at, at or above, at or below) by a given place along the course. The maneuver envelope can be split into two separate maneuver envelopes to model this situation. For example, in the lower portion of Figure 2-8, the pilot has been instructed to be at or below the indicated altitude by point P on the course. The vertices defining the envelope around the first part of this maneuver are labelled 1a, 2a, 3a, and 4a, while those defining the second envelope are labelled 1b, 2b, 3b, and 4b.

Note that very large amounts of vertical airspace may need to be protected in the absence of intermediate restrictions. However, a small number of such restrictions, even just one, can result in large reductions in the amount of airspace needing protection.

In level flight, FPCP uses as upper and lower bounds the flight altitude plus or minus a vertical threshold (a global constant with different values for altitudes below and above 29000 feet).

2.2.5 Initiating the Display of FPCP Information

When FPCP has completed its search for encounters involving the subject aircraft, the updated information is made available to the display function, including the display-as-advisory and display-as-priority times. The display function displays the advisory or priority message for an encounter to the appropriate controller(s) when the current time reaches the encounter's display-as-advisory or display-as-priority time.

2.2.6 FPCP, Sector Workload Probe and the Airspace Grid

This specification assumes that SWP and FPCP use a common airspace grid. This assumption may allow considerable savings in computer storage and execution time. The updating of the grid is a significant portion of the FPCP processing which, with a common grid, needs to be done only once per invocation of FPCP. The width of the FPCP grid cells in the horizontal and time dimensions are system parameters. There are certain implications in setting them equal to the respective horizontal and time widths of the SWP cells.

The exact horizontal dimensions of the grid cells are, within broad limits, not critical to SWP. They are critical to FPCP; the exact value is still to be determined but is expected to fall within SWP's broad limits. On the other hand, the exact dimension of each grid cell in the time dimension is, within broad limits, not critical to FPCP but is critical to SWP. Multiplied by 2^k for some positive integer k, it must equal some convenient length of time (e.g., 15 minutes, rather than, say, 13.79) over which the outputs are calculated and displayed. Further study may prove that the optimal range of the grid cell's time extent for FPCP does not include a value compatible with SWP's need for a conveniently-sized time unit. If so, grid-commonality implies that the cell's time extent must be rounded up to such a value, at the cost of more nominees and more calls to the FPCP Fine Filter. There are certain advantages, however, even apart from SWP considerations, for the FPCP grid to be divided on the time dimension in convenient clock increments, even if these are not quite optimal in reducing nominees. For instance, to do a trial probe the controller might want a list of all encounters with display-as-advisory times earlier than 3:00 p.m.

The costs associated with assuming common grids for both SWP and FPCP appear to be outweighed by the benefits. It is worth noting, however, in view of the fact that SWP and FPCP system parameter values have yet to be determined, that different grids could be used for SWP and FPCP without substantial changes in either specification. The Grid Chain Generator (Section 4.1) would simply be run twice, once for each grid, and at each trajectory update, both grids and both trees would be updated.

2.2.7 Boundary Considerations

FPCP determines conflicts throughout the planning region, which includes the center plus a buffer region. The algorithm has

been designed under the assumption that controllers have as much time to resolve conflicts transitioning into their center as they have for conflicts entirely within the center. For boundaries with non-AERA airspace, the buffer region is assumed large enough to allow the usual level of information promptness and completeness for any conflict with a point of violation within the center's airspace.

2.2.8 Controller Interface

This section discusses controller interface issues, including who may access FPCP outputs and who may trigger and/or accept a trial probe.

2.2.8.1 Who Is Informed of Conflicts

This specification does not present an algorithm for determining which controllers are notified automatically of conflicts. For FPCP to work as designed, however, the controllers who are responsible for resolving the conflicts (including those responsible for the sectors the aircraft will occupy at the display-as-priority time and perhaps the display-as-advisory time) must be notified automatically. It may be necessary to notify the controller who would be responsible if the controller(s) currently responsible fail(s) to resolve the conflict. Other controllers may benefit from seeing the conflict data upon request. It may also benefit the controllers responsible for conflicts to be able to access, upon request, data concerning encounters (which may become conflicts in the near future).

2.2.8.2 Trial Probes

This specification places no explicit constraints on who may perform trial probes. Any controller may probe any aircraft having a current trajectory. In later versions of AERA, automation functions such as Conflict Resolution and Metering may invoke a trial probe. Any number of controllers or functions may perform trial probes simultaneously, but each may test only one trial trajectory (that of its subject aircraft) at a time.

2.2.8.3 Who May Accept Trial Flight Plans

No change is anticipated for AERA 1 in the current rules for deciding which controller may give a clearance to a pilot for a new plan. These rules are discussed in the FAA's document "Air Traffic Control" [4]. Controllers must coordinate clearances with each other under AERA 1 just as they do in NAS.

3. FLIGHT PLAN CONFLICT PROBE FUNCTIONAL DESIGN

Figure 3-1 illustrates the high-level organizational structure of the Flight Plan Conflict Probe algorithm.

3.1 Environment

This section describes required input and output data, and lists conditions causing activation of the FPCP algorithm.

3.1.1 Input Data and Activation

3.1.1.1 Input Data

The input to FPCP consists of:

- The trajectory and maneuver envelopes of each aircraft in the data base, and the trial trajectory and maneuver envelopes, if any
- The stimulus for invoking FPCP (trajectory update, horizon update, trial probe)
- The identity of the subject's trajectory (trajectory update and trial probe only)
- The identity of the trial trajectory (trial probe only)

3.1.1.2 Automatic Activation Sequences

Flight Plan Conflict Probe is triggered automatically by either a trajectory update or a horizon update.

3.1.1.3 Controller Initiating Sequences

A controller may initiate a trial probe for a particular aircraft. The controller enters a trial flight plan for the aircraft into the system and its trajectory is constructed by the Trajectory Estimation Function. FPCP is invoked with this aircraft as the subject. However, the current (original) trajectory would be used should FPCP be invoked immediately thereafter for a different subject, either automatically or via another controller's trial probe.

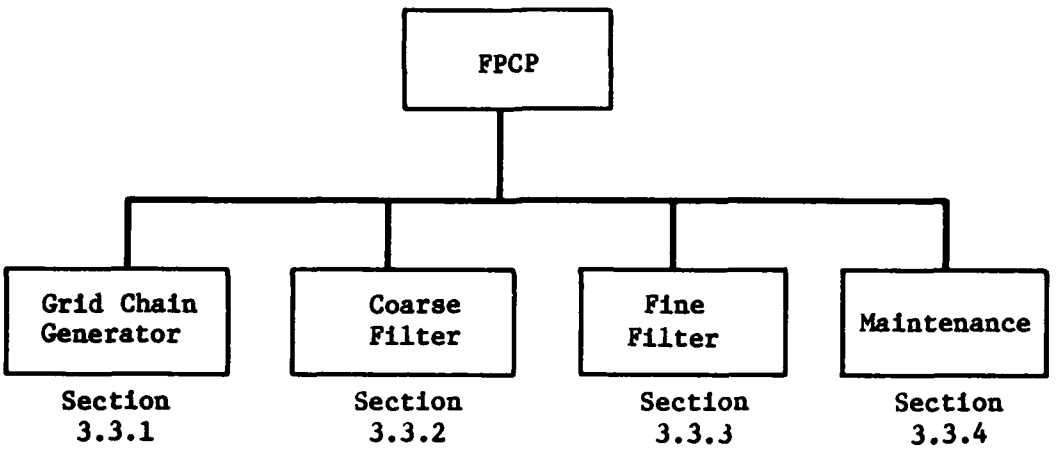


FIGURE 3-1
FPCP ORGANIZATIONAL STRUCTURE

3.1.2 Output Data

3.1.2.1 Output to Global Data Base

FPCP contributes the following to the global data base:

- A table containing information on each current encounter, including the identity of the aircraft involved and the geometry of the encounter. It is the culmination of the FPCP processing. The data are accessed by Sector Workload Probe.

- A table containing information on each encounter that has been determined no longer current (due to a trajectory update). Sector Workload Probe uses this table for housekeeping purposes, after which it deletes the table.

- A table containing information on the sparse grid chain of each object aircraft. Flight Plan Conflict Probe creates this table for determining conflicts. This table is also used by Sector Workload Probe to determine time and sectoring information for workload distribution.

FPCP makes information on encounters available to the display function (not described in this specification). The table provides enough information to determine when to display advisory and priority messages.

3.2 Design Assumptions

Three stimuli can cause an invocation of FPCP: the trajectory update, horizon update, and trial probe. The subfunctions are developed with the stimulus type as an input and use algorithms common to all three types. Any special logic required to perform an operation particular to a given stimulus can be invoked when necessary.

For a trial probe, two additional inputs must be passed to FPCP: the identity of the subject aircraft's trial trajectory, and the identity of its existing trajectory. By the time of FPCP invocation, Trajectory Estimation will have output both trajectories to the global data base. The trial trajectory's identity must differ from that of the subject (as well as from those of the other aircraft) in order to avoid determining a "conflict" of the subject with itself.

For a trajectory update, the subject's identity is input to FPCP. Again, by the time of FPCP invocation, Trajectory Estimation will have updated the subject's trajectory.

For a horizon update, FPCP requires no inputs except the stimulus type. FPCP is invoked periodically, at intervals of delta horizon. It determines the set of aircraft whose trajectories extend beyond the time horizon. The portions of the trajectories which are within a time interval of length delta horizon beyond the time horizon are processed in turn, leading to the identification of possible encounters. All of the subfunctions of FPCP, that is, the Grid Chain Generator, Coarse Filter, Fine Filter, and portions of Maintenance are invoked, in turn, for each trajectory portion.

3.3 Subfunctions

3.3.1 The Grid Chain Generator

The Grid Chain Generator (GCG) uses the cusp data from the trajectories as input data for the subject aircraft. It orders these by time into consecutive pairs to form segments. It also uses data on the maneuver envelopes to provide additional protection about the subject aircraft's maneuvers. Its outputs are the following:

- The sparse grid chain
- The buffer grid chain
- The sparse subject tree
- The buffer subject tree

3.3.1.1 The Sparse Grid Chain

The Grid Chain Generator algorithm loops through each segment of the subject's segment chain, marking certain cells it passes through as occupied (as defined in Section 2.1.7, Cell Occupancy, Grid Chains, and Buffer Cells) and adding them to the sparse grid chain. First, the cell containing the segment's (beginning) cusp is added to the grid chain. The direction (x, y, or t) in which motion through the grid is most rapid (in terms of the number of cells crossed per unit length) is designated the steepest direction.

A point on the segment is considered whose projection is one cell width farther along the axis of the steepest direction; the cell in which it lies is added to the grid chain. This cell is an orthogonal or diagonal neighbor of the previous cell. Note that the trajectory may pass through other cells

that are not marked occupied. Such cells are only "nicked" by the segment, i.e., the segment intersects only one octant.

The GCG continues adding occupied cells to the grid chain in this manner until the cusp marking the next segment is reached. For the new segment, the steepest direction is updated if it changes, and the cell occupied by the beginning cusp is added (if it is not already in the grid chain). The process continues until the last segment is reached.

Figure 3-2 illustrates an example in two dimensions. Two straight line segments joined by a cusp are shown. The corresponding occupied cells are marked by a shaded circle. Such cells compose the sparse grid chain.

3.3.1.2 Vertical Maneuvers and Holding Patterns

The sparse grid chain as derived from the trajectory information requires no further modification for trajectory segments representing straight lines or turns in level flight. However, the existence of vertical maneuvers and holding patterns necessitates the addition of extra protection about the nominal trajectory derived solely from the cusp data.

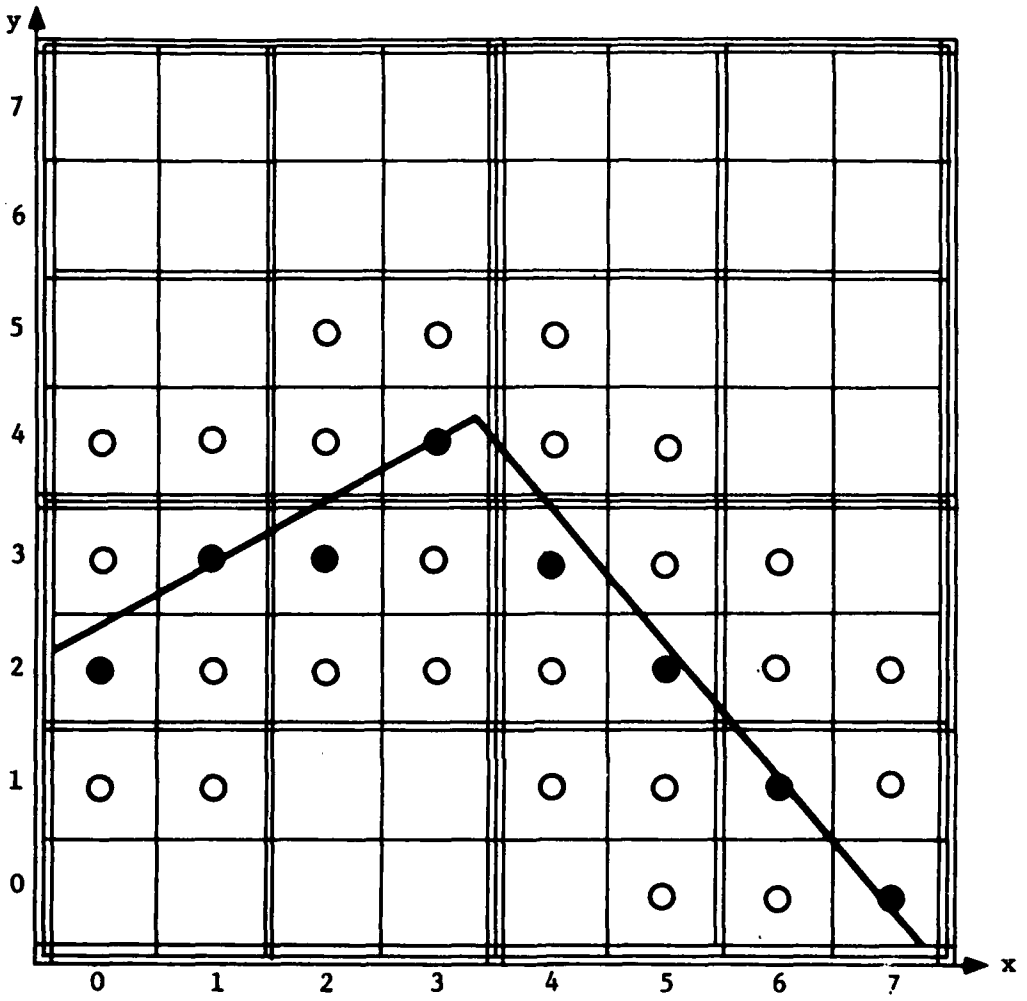
In the case of vertical maneuvers, the altitude range to be protected for each sparse cell is stored.

The case of a holding pattern may require the use of additional cells to protect against uncertainty in the aircraft's position. The cells added represent the union of (a) those cells that would be added to the sparse grid chain were each boundary edge of the geometric structure enveloping the holding pattern (in (x,y,t) space) treated in turn as a segment, and (b) any interior cells.

3.3.1.3 The Buffer Grid Chain

The buffer grid chain is a list of all cells whose x, y and t coordinates differ by no more than one cell width from a cell in the sparse grid chain. Buffer grid chain cells are shown as unshaded circles in Figure 3-2. The sparse grid chain's cells form a subset of the cells of the buffer grid chain.

It can be shown that if no cells of the buffer grid chain of a subject aircraft coincide with a cell of an object's sparse grid chain, the subject and object do not violate the FPCP separation criteria.



Legend:

● Cells in Sparse Grid Chain

○ Cells Added to Form Buffer Grid Chain

FIGURE 3-2
SPARSE AND BUFFER CELLS ASSOCIATED WITH A TRAJECTORY

3.3.1.4 Information Stored with Sparse and Buffer Cells

The grid chain includes the following information for each of its cells:

- the trajectory identifier (sparse cells only)
- a cell number called a node identifier
- the earliest and latest times of the first and last segments, respectively, occupying the cell
- the minimum and maximum altitude for which protection is provided

3.3.1.5 Generating a Tree from a Grid Chain

The last task of the Grid Chain Generator is to construct the sparse subject tree and the buffer subject tree. The trees are built from the leaves to the root. Each cell in the sparse grid chain is referenced by a leaf in the tree. The GCG stores each parent node/child node relationship for the subject. This consists of determining the unique (2x2x2) block (composed of 8 cells) containing the cell. The cells and blocks may be numbered in such a way that these relationships can be identified by bit manipulations on the cell number. A tree node is created for the block if none yet exists. A two way link is established between the cell's node (the child) and the block's node (the parent). A list is made of each 2x2x2 block that is represented in the tree. Next, the process is repeated to include in the tree the 4x4x4 blocks containing the 2x2x2 blocks, and so on, until the whole airspace (root of tree) is reached.

A temporary tree is built for the buffer grid chain in a similar way except that the subject trajectory identifier is not needed.

3.3.2 The Coarse Filter

The Coarse Filter searches each cell of the subject's buffer grid chain to see if it is also a cell in the sparse grid chain of any of the object aircraft. The algorithm outputs a nominee table, which specifies for each object aircraft which cells are co-occupied with the subject and which segments are responsible for each co-occupancy. The nominee table is passed on to the Fine Filter (Section 3.3.3) for further, more detailed tests.

In practice, it is not always necessary to search each cell in the subject's buffer grid chain. The fact that the grid chains are stored as trees allows the algorithm to terminate a search quickly if no object aircraft are in the vicinity of the subject. The algorithm compares the subject's tree with the allobject tree. An absent node in either tree indicates vacant airspace in the corresponding block of the (x,y,t) grid. If a node is absent in one or both trees, the Coarse Filter does not need to look at any descendants of the node (i.e., subsets of the node's (x,y,t) block). Appendix C develops some background on trees and the technique of recursion which is necessary in order to explain the Coarse Filter processing in more detail.

Once an object aircraft is found to co-occupy an (x,y,t) cell with the subject, the altitude test simply checks for overlap of the two ranges (subject's minimum altitude this cell, subject's maximum altitude this cell) and (object's minimum altitude this cell, object's maximum altitude this cell). When an overlap occurs, the necessary data on the subject and object segments contained in the cell are added to the nominee table.

3.3.3 The Fine Filter

The Fine Filter of Flight Plan Conflict Probe is a subfunction designed to identify all encounters (i.e., violations of FPCP separation criteria) between a single subject aircraft and an object aircraft in the nominee table provided by the Coarse Filter. It accomplishes this task by analyzing the segments of aircraft trajectories which are associated with co-occupied cells identified in the nominee table. Specifically, for every object aircraft in the nominee table, the Fine Filter performs a series of tests to determine if the subject and object aircraft segments which are associated with a co-occupied cell violate FPCP vertical and horizontal separation criteria. A special check is made to assure that no two segments are compared more than once, thus avoiding duplicate processing. If an encounter between the subject aircraft and a nominee is detected, then a table which maintains encounter information for all of the aircraft in the planning region is updated with this latest information.

3.3.3.1 Tests of Time and Space

The subject and object aircraft segment pair undergo a series of tests in time and space which indicate whether or not an encounter is possible. The tests are performed in sequence, each successive test checking for segment non-compliance with stated goals in a particular dimension. Only those segment

pairs which do not satisfy a goal are passed on to subsequent tests. All other pairs are eliminated from further consideration. Thus, it is desirable that the tests be ordered so as to eliminate as many of the non-conflicting segment pairs as possible during the earlier stages of the process. Accordingly, the segments are first checked in time. If they overlap in time, they are checked for violation of vertical separation within the time interval in common between the two segments.

The vertical separation criterion is a function of the altitudes of the aircraft. If the vertical separation criterion is violated during the common time interval, the segments are tested for violation of horizontal separation criteria. If both criteria are violated, an encounter is predicted to occur along the segments. Information relevant to each encounter, such as the minimum separation distance in the horizontal plane and the time of minimum separation, is calculated and stored in an encounter table.

3.3.3.2 The Encounter Table

The encounter table is contained in the global data base and provides information about all aircraft in the planning region that are predicted to be involved in an encounter. Within the table is stored information about every encounter detected by the temporal and spatial tests. The table contains the flight plan identifications and a set of parameters which include the start and end times of the advisory and priority violations, the display-as-advisory and display-as-priority times, the minimum separation distance of the aircraft in the horizontal plane, the time of minimum separation, and the positions (x,y,z) of the aircraft at the start and end of the advisory violation. This information is available to the AERA display function which notifies the appropriate controller(s) of possible conflicts.

3.3.4 Maintenance

Flight Plan Conflict Probe requires that up-to-date versions of all relevant data tables must be maintained. For this reason, FPCP includes as one of its principal subfunctions a procedure called Maintenance. Maintenance adds and deletes data associated with specific flight identifications from appropriate global and FPCP shared local tables. Details of the Maintenance subfunction are provided in Section 4.4, Maintenance.

4. DETAILED DESCRIPTION

The Flight Plan Conflict Probe has four subfunctions:

- Grid Chain Generator
- Coarse Filter
- Fine Filter
- Maintenance

The first three subfunctions are activated in succession as they are listed. The fourth subfunction is invoked as needed. The Grid Chain Generator preprocesses trajectory data to produce a set of cells occupied by the subject aircraft and produces a tree representing these cells. The Coarse Filter compares this set of cells for the subject against corresponding cells for the objects to determine any cells in both sets. The aircraft pairs are tested for violations of the vertical separation criterion within these co-occupied cells. For pairs with vertical violations, the Fine Filter analyzes the segments of the trajectories corresponding to the common cells. Tests are made sequentially to check if the segments overlap in time, altitude and horizontal distance. The Maintenance Subfunction updates the various local and global data structures, as required, during processing of the other subfunctions.

4.1 Grid Chain Generator

The Grid Chain Generator subfunction of FPCP is used to generate the sparse and buffer grid chains for the subject aircraft. Also, it generates the Sparse and Buffer Subject Trees to represent the cells of the respective grid chains. The Buffer Subject Tree is used by the Coarse Filter to detect co-occupied cells which indicate nominee status and the need for further checking by the Fine Filter. The Sparse Subject Tree is used by the Maintenance subfunction for insertion of the subject into the Allobject Tree and is kept for future use by Maintenance.

The Grid Chain Generator consists of three major components: a Sparse Cell Generator to generate a sparse grid chain for the subject trajectory, a Buffer Cell Generator to generate a buffer grid chain for the subject, and the Grid To Tree Converter to convert a sparse or buffer grid chain to a Sparse or Buffer Subject Tree.

The organizational structure of the Grid Chain Generator is illustrated in Figure 4-1.

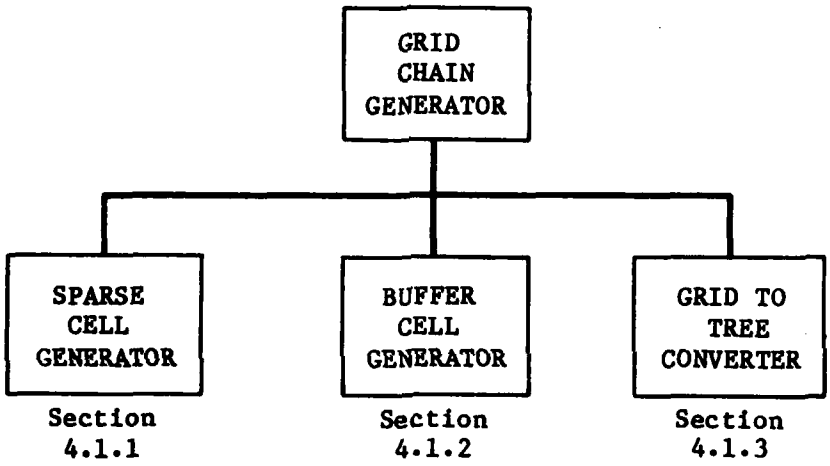


FIGURE 4-1
GRID CHAIN GENERATOR ORGANIZATIONAL STRUCTURE

4.1.1 Sparse Cell Generator

4.1.1.1 Mission

The Sparse Cell Generator component of the Grid Chain Generator updates a global table listing all cells occupied in the sparse sense by the current subject aircraft trajectory. It accomplishes this objective by using data that describes the subject's nominal trajectory in (x,y,z,t) space. It also sets upper and lower altitude protection limits for each cell associated with a vertical maneuver or holding pattern segment.

4.1.1.2 Design Considerations and Component Environment

Input

This component requires, as local inputs, the unique identifier for the subject aircraft flight plan (Subject_Fl_Id), the start time for the evaluation (T_Start), and the end time of the evaluation (T_Horizon). Usually, T_Start will be the present time, but for a horizon update it will be the old time horizon less one cell time interval, and for a trial probe it will be one time cell less than the time of divergence from the current trajectory. The input T_Horizon is the current time horizon. These inputs are provided by the Grid Chain Generator component.

Sparse Cell Generator also uses information from the global table TRAJECTORIES which describes the position and time of each trajectory cusp, as well as the type of maneuver envelope (if any) associated with each cusp.

Information on maneuver envelopes is provided by the global table MANEUVER_ENVELOPES and is used by the elements of Sparse Cell Generator. The table, SPARSE_CELLS, which is updated by the elements of Sparse Cell Generator, is also a global input.

Output

The output of Sparse Cell Generator is an updated version of the global table SPARSE_CELLS. This table contains information which is calculated by the elements of Sparse Cell Generator. It contains a list of cells that are occupied in the sparse sense by the subject aircraft's trajectory.

4.1.1.3 Component Design Logic

The Sparse Cell Generator uses a number of elements in fulfilling its goals. The organization and calling sequence of the Sparse Cell Generator is given as follows:

```
Sparse Cell Generator
   Determine Independent Variable
   Straight Line Generator
      Encode
   Vertical Protect
   Hold Area Protect
      Determine Independent Variable
      Straight Line Generator
           Encode
```

The processing logic of Sparse Cells Generator is given by the PDL in Figure 4-2. First, this component locates all TRAJECTORY records pertaining to the subject aircraft trajectory. It then transfers the required data from that table to local arrays which are ordered by time. A count of the records retrieved is noted.

Consecutive pairs of cusps are taken together to form trajectory segments. Sparse Cell Generator's main loop is repeated for all cusp pairs starting with the segment that bridges the starting time (or the first segment, if none do) and ending with the segment that bridges the time horizon (or the last segment, if none do).

The coordinate data and the time data for the cusp pairs is operated on by the elements Determine Independent Variable and Straight Line Generator. These elements are responsible for translating each segment into a sparse chain of occupied cells for the trajectory's regular segments. All segments are processed in this manner before processing begins for maneuver envelopes. This order is followed because altitude limits for cells occupied by the segments which follow the envelope may be affected by the vertical maneuver. These cells must be selected before attempting to calculate their altitude limits.

Following the creation of the sparse cells for the nominal trajectory, the Sparse Cell Generator loops on all of the segments searching for maneuver envelopes. If a vertical maneuver is found, then the Vertical Protect element is called. If a holding pattern is found (with or without altitude change), then the Hold Area Protect element is invoked.

```
ROUTINE Sparse_Cell_Generator;
#Component of Grid Chain Generator#
PARAMETERS Subject_Fl_Id IN, T_Start IN, T_Horizon IN;
REFER TO GLOBAL TRAJECTORIES IN;
DEFINE CONSTANTS
   Flag  Flag set to 1 indicates that Straight_Line_Generator
         is called from this routine to produce the nominal
         trajectory;
DEFINE VARIABLES
   Subject_Fl_Id  Flight identifier of the subject aircraft
   T_Start        Starting time for sparse cell generator
      #This is set to the present time for new flight plans and #
         #resynchronization, to the time point of the alternate#
         #trajectory for a trial probe, and to the old time horizon#
         #minus one time cell for a horizon update#
   T_Horizon                 Time horizon
   Time(*)                   The time values of the trajectory cusps
                             ordered by time
   Cell_Line_Parameters(6)   Pass through vehicle for parameters from
                                Determine_Independent_Variable to
                                Straight_Line_Generator (includes xyt
                                starting cell numbers and slopes)
   X(*)                      The x values of the trajectory cusps
                                ordered by the time of the cusps
   Y(*)                      The y values of the trajectory cusps
                                ordered by the time of the cusps
   Z(*)                      Altitudes ordered by the time of the cusp
   Cusp_Type(*)              The cusp types of the trajectory cusps
                                ordered by the time of the cusps
   M_Count                   The number of cells traversed by the
                                segment
   N_Beg                     Starting value for maneuver envelope loop
   N_End                     Ending value for maneuver envelope loop
   N_Count                   A count of trajectory records
   N                         Index to loop on cusps;
DEFINE TABLES
   REGULAR_CELLS     Table containing information on cells occupied
                        by the regular segments
      time              The nominal time at which the cell was
                           occupied
      node-id           The cell number;
```

FIGURE 4-2
SPARSE_CELL_GENERATOR

```
    SELECT FIELDS time, x, y, z, cusp_type
       FROM TRAJECTORIES (TJ)
       INTO Time, X, Y, Z, Cusp_Type
       WHERE TJ.fl_id EQ Subject_Fl_Id AND (TJ.time GT T_Start AND
          TJ.time LT T_Horizon)
       ORDERED BY TRAJECTORIES.time
       RETURN COUNT (N_Count);
    N_Beg = N_Count;
    N_End = 0;
    FOR N = 1 TO N_Count - 1;
       #determine if any portion of the segment is within the time#
          #bound of the probe#
       IF Time(N+1) GT T_Start AND Time(N) LT T_Horizon
       THEN #Find regular segment#
          #determine direction in which motion is most rapid#
          CALL Determine_Independent_Variable (Time(N+1) IN,
             X(N+1) IN, Y(N+1) IN, Time(N) IN, X(N) IN, Y(N) IN,
             Cell_Line_Parameters OUT, M_Count OUT);
             #generate cells for regular segment#
          CALL Straight_Line_Generator (Time(N+1) IN, Z(N+1) IN,
             Time(N) IN, Z(N) IN, Subject_Fl_Id IN, Flag IN,
             Cell_Line_Parameters IN, M_Count IN, REGULAR_CELLS OUT);
          N_Beg = MIN(N, N_Beg);
          N_End = MAX(N, N_End);
    FOR N = N_Beg TO N_End;# Loop for maneuver envelopes#
       IF Cusp_Type(N) EQ 'Vertical maneuver'
       THEN
          #add altitude range to cells for vertical maneuvers#
          CALL Vertical_Protect (Time(N) IN, Subject_Fl_Id IN,
             REGULAR_CELLS IN);
       IF (Cusp_Type(N) EQ 'Hold') OR (Cusp_Type(N) EQ
          'Vertical hold')
       THEN
          #add additional cells to account for boundary of hold#
          CALL Hold_Area_Protect (Time(N) IN, Subject_Fl_Id IN);
END Sparse_Cell_Generator;
```

FIGURE 4-2
SPARSE_CELL_GENERATOR (Concluded)

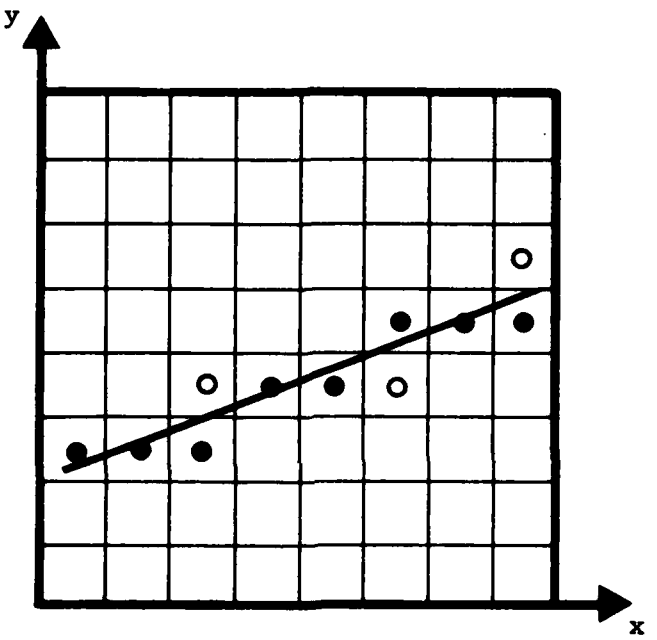
Determine Independent Variable

The element Determine Independent Variable determines in which dimension the trajectory is changing most rapidly. The variable or axis corresponding to this dimension is called the independent variable or independent axis. The following discussion describes the role of independent variables in finding the cells which are occupied in the sparse sense.

It already has been noted in Section 2.1.7 that penetration of a cell does not imply that the cell will be declared occupied. Operationally, Section 3.3.1.1, The Sparse Grid Chain, defines a cell to be occupied if a line traversing it occupies more than one octant of the (three-dimensional) cell (or quadrant of a cell in the two-dimensional example of Figure 3-2). How then is this operational condition to be implemented algorithmically? This volume does not attempt an explanation of the mathematics; instead, illustrative examples are given of how the correct grid chain is generated by using the correct independent variable, but not by using the incorrect independent variable.

Figure 4-3 shows a straight line transversing a two dimensional grid where x is the independent variable. Using the "more than one quadrant" criterion, the cells having solid circles should be identified while cells penetrated for just one quadrant (open circles) should not. Each vertical "column" of cells along the x-axis is considered in turn. The y-coordinate is computed for the point on the segment whose x-coordinate (the independent variable) is the center of the current column of cells. The cell containing the resulting y value is marked occupied. Exactly one cell will be marked occupied for each column of cells. For each horizontal "row" of cells (along the non-independent y-axis), more than one cell may be marked occupied.

Figure 4-4 shows a second straight line to which the "more than one quadrant" criterion has been applied. Cells marked with solid circles, both starred and plain, would be marked occupied. Applying the procedure as described above (with x chosen as the independent variable) will produce only the starred cells. The resultant grid chain (if passed as is--with gaps--to the Coarse and Fine Filters after applying the simple buffering scheme described in Section 4.1.2) will not provide reliable separation assurance.

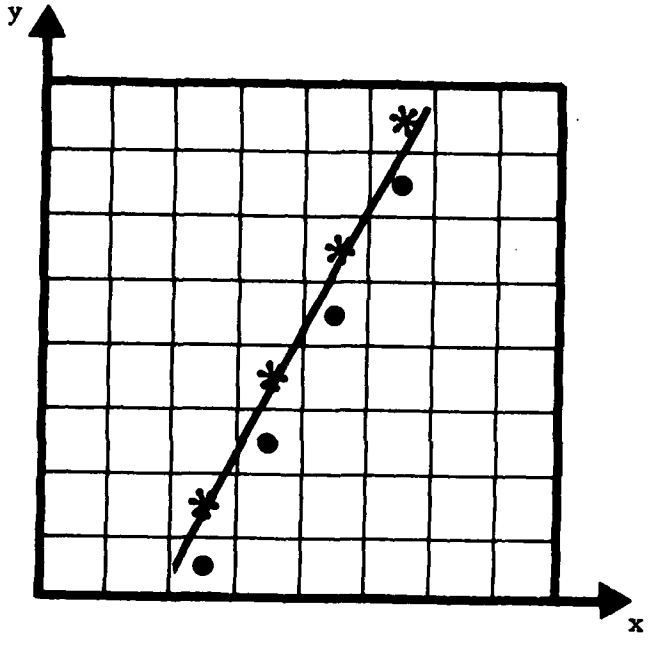


LEGEND:

● Occupied cell

○ Penetrated but unoccupied cell

FIGURE 4-3
CELL OCCUPANCIES USING CORRECT INDEPENDENT AXIS



LEGEND:

* Occupied cell based on improper choice of independent axis

● Cells needed to complete sparse grid chain

FIGURE 4-4
CELL OCCUPANCIES USING INCORRECT INDEPENDENT AXIS

A solution to this problem is to interchange the roles of x and y (and of "columns" and "rows" of cells) to make y the independent variable. Thus, in Figure 4-4, values of y at the center of a row's cells are input to the equation of the line and the resultant value of x determines which cell is marked occupied. This change in independent variable must take place when the net change in y exceeds the net change in x; that is, when the slope exceeds one.

The scheme is readily extended to three dimensions as is done in Determine Independent Variable. The PDL for this element is given in Figure 4-5. First, the coordinates of the beginning cusp of the current segment are transformed to cell coordinates. This is a linear transformation, accomplished by dividing the geographic or time coordinates by the cell size. Then an offset is added such that the integer part of the cell coordinate at the lower bound of the cell corresponds to the cell's grid number along that axis.

The length of the segment, projected along each cell coordinate axis, is computed and expressed in cell units. The absolute magnitude of these lengths (called extents) are then compared to one another and the axis associated with the greatest extent becomes the independent variable.

Finally, M_Count, the number of grid divisions traversed by the independent variable, is computed, as well as the slopes of the dependent variable with respect to the independent variable. The cell coordinates of the segment starting point, the slopes for each of the three coordinates, and the number of grid divisions traversed by the independent variable are returned to the Sparse Cell Generator for use by the element Straight Line Generator.

The PDL representation of this element is given in Figure 4-6.

Straight Line Generator

Straight Line Generator uses the output developed by the element Determine Independent Variable. In addition, Straight Line Generator uses the time coordinates for both nodes (which serve as keys for the end points of the current segment), the beginning and ending altitude for the segment, and a flag value indicating whether the Straight Line Generator is being used to service a regular segment or a holding pattern maneuver envelope.

```
ROUTINE Determine_Independent_Variable;
PARAMETERS T_End IN, X_End IN, Y_End IN, T_Beg IN, X_Beg IN, Y_Beg
   IN, T_Beg_Cell OUT, X_Beg_Cell OUT, Y_Beg_Cell OUT, T_Slope OUT,
   X_Slope OUT, Y_Slope OUT, M_Count OUT;
REFER TO SHARED LOCAL T_Cell_Dimension IN, H_Cell_Dimension IN,
   T_Offset IN, X_Offset IN, Y_Offset IN;
DEFINE VARIABLES
   T_End            Ending value for time
   X_End            Ending value for x
   Y_End            Ending value for y
   T_Beg            Beginning value for time
   X_Beg            Beginning value for x
   Y_Beg            Beginning value for y
   T_Beg_Cell       Beginning value of time expressed in cell
                       coordinates
   Y_Beg_Cell       Beginning value of y expressed in cell
                       coordinates
   X_Beg_Cell       Beginning value of y expressed in cell
                       coordinates
   T_Extent         Difference between beginning and ending values
                       of time expressed in cell coordinates
   Y_Extent         Difference between beginning and ending values
                       of y expressed in cell coordinates
   X_Extent         Difference between beginning and ending values
                       of x expressed in cell coordinates
   M_Extent         Maximum extent
   M_Count          Number of cells traversed by the segment
   T_Slope          Change in time per unit change in independent
                       variable
   Y_Slope          Change in y per unit change in independent
                       variable
   X_Slope          Change in x per unit change in independent
                       variable;
```

FIGURE 4-5
DETERMINE_INDEPENDENT_VARIABLE

```
    #Compute starting cell coordinates but do not quantize yet#
    T_Beg_Cell = T_Beg/T_Cell_Dimension + T_Offset;
    Y_Beg_Cell = Y_Beg/H_Cell_Dimension + Y_Offset;
    X_Beg_Cell = X_Beg/H_Cell_Dimension + X_Offset;
    #Compute extents as the difference in beginning cell coordinate#
       #and ending cell coordinate#
    T_Extent = (T_Beg - T_End)/T_Cell_Dimension;
    Y_Extent = (Y_Beg - Y_End)/H_Cell_Dimension;
    X_Extent = (X_Beg - X_End)/H_Cell_Dimension;
    #Determine independent variable as one having greatest extent#
    #Compute the count of cells along that axis#
    CHOOSE CASE #M_Extent defines the independent variable#
       WHEN (ABS(T_Extent) GE ABS(X_Extent)) AND (ABS(T_Extent)
          GE ABS(Y_Extent)) THEN
          M_Extent = T_Extent;
          M_Count = CEIL(T_Beg_Cell) - CEIL(T_End/T_Cell_Dimension +
             T_Offset);
       WHEN (ABS(Y_Extent) GE ABS(X_Extent)) THEN
          M_Extent = Y_Extent;
          M_Count = CEIL(Y_Beg_Cell) - CEIL(Y_End/H_Cell_Dimension +
             Y_Offset);
       OTHERWISE
          M_Extent = X_Extent;
          M_Count = CEIL(X_Beg_Cell) - CEIL(X_End/H_Cell_Dimension +
             X_Offset);
    #Compute slopes of each variable with respect to independent#
       #variable#
    T_Slope = T_Extent/M_Extent;
    Y_Slope = Y_Extent/M_Extent;
    X_Slope = X_Extent/M_Extent;
END Determine_Independent_Variable;
```

FIGURE 4-5
DETERMINE_INDEPENDENT_VARIABLE (Concluded)

```
ROUTINE Straight_Line_Generator;
PARAMETERS T_Exit IN, Z_End IN, T_Entry IN, Z_Beg IN, Subj_Fl_Id IN,
   T_Beg_Cell IN, Flag IN, T_Beg_Cell IN, X_Beg_Cell IN, Y_Beg_Cell
   IN, T_Slope IN, X_Slope IN, Y_Slope IN, M_Count IN, REGULAR_CELLS
   OUT;
REFER TO GLOBAL SPARSE_CELLS INOUT;
REFER TO SHARED LOCAL T_Cell_Dimension IN, T_Offset IN;
DEFINE VARIABLES
   T_Entry          Entry time for current segment
   T_Exit           Exit time for current segment
   Z_End            Ending value for altitude
   Z_Beg            Beginning value for altitude
   Flag             Set to 1 when routine is called to find regular
                       segment, 0 otherwise
   T_Beg_Cell       Beginning value of time expressed in cell
                       coordinates
   Y_Beg_Cell       Beginning value of y expressed in cell coordinates
   X_Beg_Cell       Beginning value of x expressed in cell coordinates
   T_Slope          Change in time per unit change in independent
                       variable
   Y_Slope          Change in y per unit change in independent
                       variable
   X_Slope          Change in x per unit change in independent
                       variable
   Subject_Fl_Id    The flight id of the current subject
   T_Cell           Time expressed in cell coordinates
   Y_Cell           Y expressed in cell coordinates
   X_Cell           X expressed in cell coordinates
   Temp_T           Time at which the regular segment passes through
                       center of cell
   Z_Up             Altitude of regular segment at center of cell
   Z_Down           Altitude of regular segment at center of cell
   Temp_Node        Temporary node id
   M_Count          Number of cells traversed by segment
   M                Counting index;
DEFINE TABLES
   REGULAR_CELLS    Information on cells occupied by regular segment
      time             Time for cell
      node_id          Cell number;
```

FIGURE 4-6
STRAIGHT_LINE_GENERATOR

```
    # Loop through cells along independent variable axis #
    FOR M = 0 TO M_Count;
       T_Cell = CEIL(T_Beg_Cell + T_Slope*M - 0.5);
       Y_Cell = CEIL(Y_Beg_Cell + Y_Slope*M - 0.5);
       X_Cell = CEIL(X_Beg_Cell + X_Slope*M - 0.5);
       #Convert the cell grid coordinates to a Node number#
       CALL Encode(T_CELL IN, Y_Cell IN, X_Cell IN, Temp_Node OUT);
       Temp_T = (T_Cell - T_Offset - 0.5)*T_Cell_Dimension;
       Z_Up = Z_Beg;
       Z_Down = Z_End;
       IF (Z_End NE Z_Beg) AND (Flag EQ 1) #altitude is changing and #
          #doing a regular segment#
       THEN
          #interpolate to find altitude#
          Z_Up = Z_Beg + (Temp_T - T_Beg)*(Z_End - Z_Beg)/
             (T_End - T_Beg);
          Z_Down = Z_Up;
       IF COUNT((SPARSE_CELLS.tree_node_id EQ Temp_Node) AND
          (SPARSE_CELLS.fl_id EQ Subject_Fl_Id)) NE 0
       THEN
          UPDATE IN SPARSE_CELLS (min_z = MIN(min_z, Z_Down),
             max_z = MAX(max_z, Z_Up), entry_time = MIN(entry_time,
             T_Beg), exit_time = MAX(exit_time, T_End))
             WHERE SPARSE_CELLS.tree_node_id EQ Temp_Node AND
                SPARSE_CELLS.fl_id EQ Subject_Fl_Id;
       ELSE
          INSERT INTO SPARSE_CELLS (fl_id = Subject_Fl_Id,
             tree_node_id = Temp_Node, min_z = Z_Down, max_z = Z_Up,
             entry_time = T_Beg, exit_time = T_End);
       IF Flag EQ 1 #routine called to produce regular segment#
       THEN
          INSERT INTO REGULAR_CELLS (time = Temp_T, node_id =
             Temp_Node);
END Straight_Line_Generator;
```

FIGURE 4-6
STRAIGHT_LINE_GENERATOR (Concluded)

Straight Line Generator then uses this information to compute cell occupancy. It does this by iterating over unit increments of the independent variable. Using the equation for a straight line, the position of the two dependent variables is computed as a formation of the independent variable and expressed in cell coordinates. (The independent variable is also recomputed, but since its slope is one, its computation returns the original value.) All of the coordinates are then truncated to integer values (using the CEIL function) which gives the grid coordinates of each occupied cell.

For purposes of cell identification and future manipulations via recursive algorithms, it is convenient to merge the three cell coordinates into a single identifier called the tree node identifier or node identifier for short. This is done by the utility element, Encode. There are many ways of encoding the cell coordinates into a single identifier. One such encoding method is particularly natural and is presented as an example. It helps illustrate the concepts involved and the close relationships existing between the grid cells and blocks and the corresponding tree nodes.

Figure 2-4 shows a trajectory crossing an 8 by 8 grid and the tree structure that applies to that trajectory. Each leaf is characterized by a three-digit base four number. This version of Encode takes two coordinates and converts them to the leaf number.

Figure 4-7 illustrates the conversion for a particular cell (labeled 022) in Figure 2-4 (the leftmost cell of the trajectory). The coordinates of that cell are x = 0 (first column) and y = 3 (fourth row from bottom). The binary expressions of these coordinates (x=000, y=011) are shown at the top of Figure 4-7. Shuffling the bits as shown results in the tree node identifier, 022. This is indeed an occupied leaf-level node as can be seen in Figure 2-4.

In this same loop, Straight Line Generator derives the effective time (Temp_T) of each cell. This is the time at which the independent variable traverses the center of the cell. It is used for two purposes:

- computing the altitude using linear interpolation methods
- incorporating effective time for regular segments in a table called REGULAR_CELLS. (This is needed to process vertical maneuver envelopes)

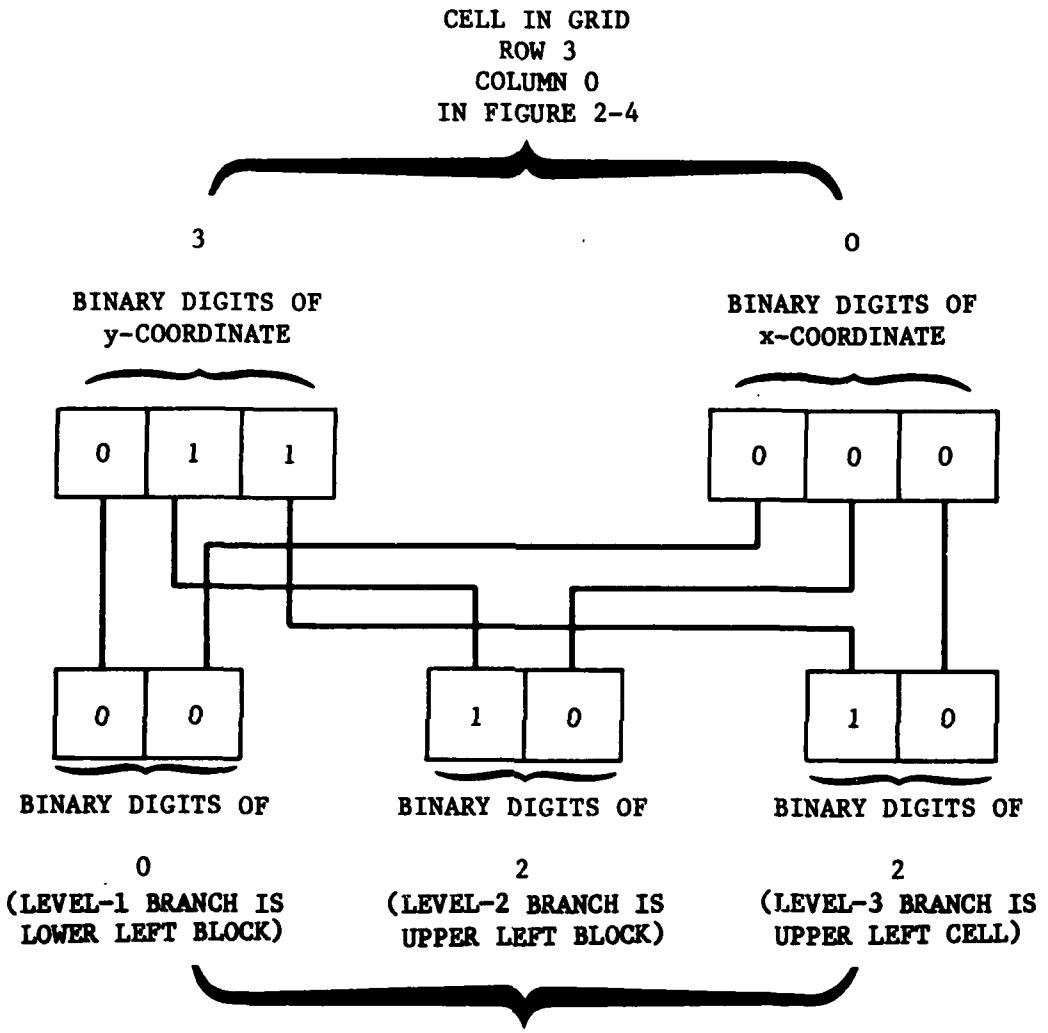


FIGURE 4-7
AN EXAMPLE FOR A METHOD OF CONVERTING FROM GRID CELL COORDINATES TO A TREE NODE IDENTIFIER

Finally, the outputs of the Straight Line Generator element as called from Sparse Cell Generator are computed as follows: if the cell is new, then a record is inserted into SPARSE_CELLS including the following fields

- The subject flight identifier
- The minimum and maximum altitude (which have the same value for regular segments)
- The entry and exit times for the cell

If the cell exists from previous computations as indicated by a duplication of the flight identifier and the node identifier, then the other fields are updated as follows:

- The minimum and maximum altitudes are the minimum and maximum of the old and current altitudes
- The entry time is the minimum of current segment beginning time or previous entry time
- The exit time is the maximum of the current segment ending time or the previous exit time

The Determine Independent Variable and Straight Line Generator processing has been applied to the entire trajectory shown in Figure 3-2, and the results are partially shown in Table 4-1. From left to right, the table shows, for each cell, the following:

- The cell coordinates
- The corresponding identifiers (using the scheme illustrated in Figure 4-7)
- The entry time and exit time
- Additional comments

Vertical Protect

If the current segment being processed by the Sparse Cell Generator is associated with a vertical maneuver, then the minimum and maximum altitudes are not identical. The element Vertical Protect determines their values. The PDL representation of this element is shown in Figure 4-8.

TABLE 4-1
EXAMPLE SPARSE CELL TABLE

Cell Coordinates		CELL DATA			
		Node	Entry	Exit	Comment
y	x	ID	Time	Time	
2	0	020	t_1	t_2	Entry into planning region at t_1
3	1	023	t_1	t_2	
3	2	032	t_1	t_2	
4	3	211	t_1	t_3	Cusp at t_2
3	4	122	t_2	t_3	
2	5	121	t_2	t_3	
1	6	112	t_2	t_3	
0	7	111	t_2	t_3	Exit from planning region at t_3

```
ROUTINE Vertical_Protect;
PARAMETERS Beg_Time IN, Subject_Fl_Id IN, Regular_Cells IN;
REFER TO GLOBAL MANEUVER_ENVELOPES IN, SPARSE_CELLS INOUT;
DEFINE VARIABLES
   Subject_Fl_Id  The flight id of the current subject
   Beg_Time       The time at first cusp in the current segment
   Zru            Current altitude
   Tru            Latest time aircraft can leave current altitude
   Zlu            Current altitude
   Tlu            Earliest time aircraft can leave current altitude
   Zld            Target altitude
   Tld            Earliest time aircraft can arrive at target
                     altitude
   Zrd            Target altitude
   Trd            Latest time aircraft can arrive at target altitude
   Slope_L        Left slope for vertical maneuver
   Slope_R        Right slope for vertical maneuver
   Temp_T         Temporary time variable
   Temp_Node      Temporary node ID
   Z_Entry        Altitude at entry to maneuver envelope
   Z_Exit         Altitude at exit from maneuver envelope
DEFINE TABLE
   REGULAR_CELLS  Like REGULAR_CELLS in SPARSE_CELL_GENERATOR
```

FIGURE 4-8
VERTICAL_PROTECT

```
    SELECT FIELDS rd_z, rd_t, ru_z, ru_t, lu_z, lu_t, ld_z, ld_t
       FROM MANEUVER_ENVELOPE (ME)
       INTO Zrd, Trd, Zru, Tru, Zlu, Tlu, Zld, Tld
       WHERE ME.fl_id EQ Subject_Fl_Id AND ME.time EQ Beg_Time;
       Slope_R = (Zru - Zrd)/(Tru - Trd);
       Slope_L = (Zlu - Zld)/(Tlu - Tld);
       #Compute altitudes for the upper and lower limits of maneuver#
          #envelope#
       REPEAT FOR EACH REGULAR_CELLS RECORD
          WHERE REGULAR_CELLS.time GT Tlu
             AND REGULAR_CELLS.time LT Trd;
          Temp_T = REGULAR_CELLS.time;
          Z_Entry = Zlu;
          Z_Exit = Zrd;
          IF Temp_T GT Tru
          THEN #time within bounds of maneuver envelope, interpolate#
             #to find exit time#
             Z_Exit = Z_Exit + (Temp_T - Tru)*Slope_R;
          IF Temp_T LT Tru
          THEN #time within bounds of maneuver envelope, interpolate#
             #to find entry time#
             Z_Entry = Z_Entry + (Temp_T - Tld)*Slope_L;
          Temp_Node = REGULAR_CELLS.node;
          UPDATE IN SPARSE_CELLS (min_z = MIN(min_z, Z_Entry, Z_Exit),
             max_z = MAX(max_z, Z_Entry, Z_Exit))
             WHERE SPARSE_CELLS.fl_id EQ Subject_Fl_Id AND
                SPARSE_CELLS.tree_node_id EQ Temp_Node;
END Vertical_Protect;
```

FIGURE 4-8
VERTICAL_PROTECT (Concluded)

The two inputs, the time of the cusp and the subject trajectory identifier, are used to select the appropriate record from MANEUVER_ENVELOPES. Only the z and t fields are extracted. The x and y fields are not used. (Indeed, they may have no physical meaning. Trajectory Estimation computes the four vertices of a cusp's maneuver envelope before considering later cusps. A later cusp may nevertheless occur prior to some of the vertices. If such a cusp represents a turn, the x and y coordinates of those vertices will not reflect the turn. The z and t coordinates, however, are valid.)

Altitudes for the upper and lower limits of the maneuver envelope are computed by linear interpolation. This is done by looping on all cells occupied by the segment (stored in the local table REGULAR_CELLS) that fall within the timeframe of the maneuver. Then the node identifier for each regular cell is used to find the applicable SPARSE_CELLS record that needs to be updated.

Updating the altitude range in the table SPARSE_CELLS is effected as follows: The maximum altitude is the maximum of (a) the old maximum altitude, (b) the upstream slope or "left" side of the envelope as applicable (see Figure 2-8), or (c) the downstream slope or "right" side of the envelope as applicable. Similarly the minimum slope is the minimum of the old minimum or of (b) or (c) above.

Hold Area Protect

If the current segment being processed by the Sparse Cell Generator is a hold, then the volume in (x,y,t) defined by the holding pattern must be protected. This is done by the element Hold Area Protect. The PDL representation for this element is shown in Figure 4-9.

Two inputs, the subject's flight identifier and the time of the cusp, are used to select the appropriate record from the global table MANEUVER_ENVELOPES.

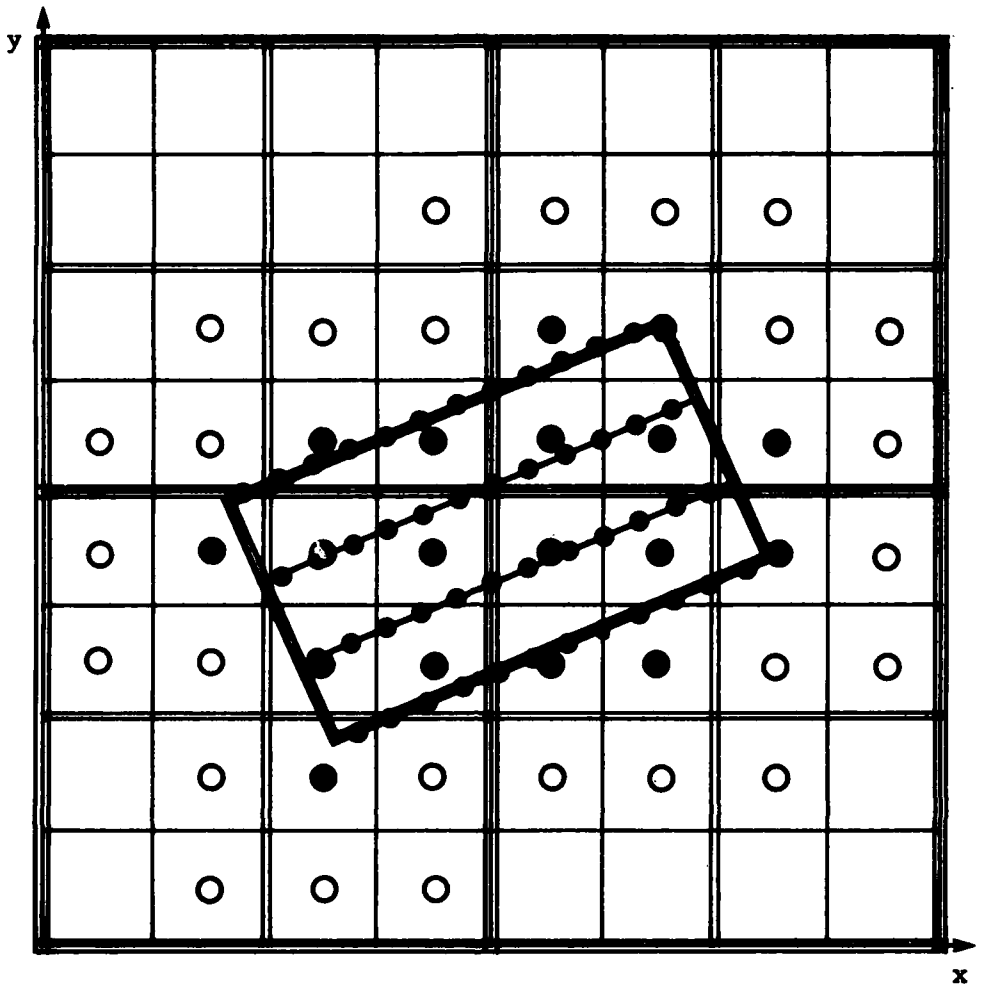
The outside loop in Hold Area Protect generates a complete set of hold cells in (x,y) space for each time unit (expressed in cell coordinates) for which the hold is planned. The inside loop creates a set of straight line segments parallel to the right side of the hold, and separated by no more than one horizontal cell width, as shown in Figure 4-10. Hold Area Protect generates the end points of these lines and uses the Determine Independent Variable and the Straight Line Generator

```
ROUTINE Hold_Area_Protect;
PARAMETERS Time IN, Subject_Fl_Id IN;
REFER TO GLOBAL MANEUVER_ENVELOPES IN;
REFER TO SHARED LOCAL T_Cell_Dimension IN, H_Cell_Dimension IN,
   T_Offset IN;
DEFINE CONSTANTS
   Flag              Set equal to 0 To indicate that Straight_Line_
                        Generator is called from this routine to
                        update the table of sparse cells only;
DEFINE VARIABLES
   Time              Time key of the hold maneuver envelope
   Subject_Fl_Id     Current subject flight identifier
   Xrd,Yrd,Zrd,Trd   Coordinates of the right downstream vertex
   Xru,Yru,Zru,Tru   Coordinates of the right upstream vertex
   Xlu,Ylu,Zlu,Tlu   Coordinates of the left upstream vertex
   Xld,Yld,Zld,Tld   Coordinates of the left downstream vertex
   T_Beg             Start time of hold
   T_End             End time of hold
   Z_Max             Highest altitude of hold
   Z_Min             Lowest altitude of hold
   Cell_Line_        Pass through vehicle for parameters
      Parameters(6)
   M_Count           Number of cells traversed by the segment
   M_Extent          Number of time cells for time cell loop
   X_Extent          Length of x component of downstream side of
                        hold
   Y_Extent          Distance of y component of downstream side
                        of hold
   N_Extent          Number of parallel straight line segments
                        needed to cover hold
   X_Delta           X increment for locations and points of
                        parallel lines
   Y_Delta           Y increment for locations and points of
                        parallel lines
   T_Temp            Current effective time for cells in hold
                        maneuver
   M                 Loop index
   N                 Loop index
   X_End             Final x coordinate for line covering hold
   X_Beg             Starting x coordinate for line covering hold
   Y_End             Final y coordinate for line covering hold
   Y_Beg             Starting y coordinate for line covering hold;
DEFINE TABLES
   DUMMY             Dummy table defined like REGULAR_CELLS
```

FIGURE 4-9
HOLD_AREA_PROTECT

```
SELECT FIELDS  right_downstream_vertex, right_upstream_vertex,
   left_upstream_vertex, left_downstream_vertex
   FROM MANEUVER_ENVELOPES (ME)
   INTO Xrd, Yrd, Zrd, Trd, Xru, Yru, Zru, Tru,
      Xlu, Ylu, Zlu, Tlu, Xld, Yld, Zld, Tld
   WHERE ME.fl_id Eq Subject_Fl_id AND ME.time EQ Time;
#Find start and end times of hold#
T_Beg = MIN(Trd, Tru, Tlu, Tld);
T_End = MAX(Trd, Tru, Tlu, Tld);
#Find highest and lowest altitudes in hold#
Z_Max = MAX(Zrd, Zru, Zlu, Zld);
Z_Min = MIN(Zrd, Zru, Zlu, Zld);
#Find number of cells corresponding to the hold duration#
M_Extent = CEIL(T_Beg/T_Cell_Dimension + T_Offset) -
   CEIL(T_End/T_Cell_Dimension + T_Offset);
X_Extent = Xrd - Xld;
Y_Extent = Yrd - Yld;
#Find independent variable along downstream side and set N_Extent#
#to twice the maximum number of cell sized steps along that side #
IF ABS(Y_Extent) GE ABS(X_Extent)
THEN
   N_Extent = 2 * CEIL(Y_Extent/H_Cell_Dimension);
ELSE
   N_Extent = 2 * CEIL(X_Extent/H_Cell_Dimension);
T_Temp = T_Beg;
#Find increments along downstream side for spacing of parallel#
#lines to cover hold                                          #
X_Delta = X_Extent/N_Extent;
Y_Delta = Y_Extent/N_Extent;
FOR M = 0 TO M_Extent; #Step through the hold along the time axis#
   T_Temp = T_Temp + T_Cell_Dimension;
   FOR N = 0 TO N_Extent; #Step along the downstream side#
      X_End = Xld + X_Delta*N;
      X_Beg = X_End - Xld + Xlu;
      Y_End = Yld + Y_Delta*N;
      Y_Beg = Y_End - Yld + Ylu;
      CALL Determine_Independent_Variable (T_Temp IN, X_End IN,
         Y_End IN, T_Temp IN, X_End IN, Y_End IN, Cell_Line_
         Parameters OUT, M_Count OUT);
      CALL Straight_Line_Generator (T_End IN, Z_Min IN, T_Beg IN,
         Z_Max IN, Subj_Fl_Id IN, Flag IN, Cell_Line_Parameters
         IN, M_Count IN, DUMMY OUT);
END Hold_Area_Protect;
```

FIGURE 4-9
HOLD_AREA_PROTECT (Concluded)



LEGEND:

Boundary of Hold

Lines Input to Straight Line Generator

Sparse Cells Designated by Straight Line Generator

Buffer Cell

FIGURE 4-10
OCCUPIED CELLS FOR A HOLDING PATTERN

elements to update or insert additional records in the global table SPARSE_CELLS.

4.1.2 Buffer Cell Generator

4.1.2.1 Mission

The Buffer Cell Generator component of the Grid Chain Generator outputs a shared local table of buffer cells using the sparse cells of the current subject. When these buffer cells are compared to the sparse cells of any other trajectory, violations of separation can be ruled out if no overlap occurs. It is sufficient that buffer cells include sparse cells and all nearest diagonal and orthogonal neighbors of the sparse cells to achieve separation assurance.

In addition to buffering with respect to cells, Buffer Cell Generator determines vertical separation minima and maxima for each buffer cell. It allows the identification of all pertinent trajectory segments (used by the Fine Filter subfunction of Flight Plan Conflict Probe).

4.1.2.2 Design Considerations and Component Environment

Input

The input to Buffer Cell Generator includes the global table SPARSE_CELLS, the subject aircraft's flight identifier, the current time, and the time horizon.

Output

The output is the shared local table BUFFER_CELLS. Each buffer cell, like each sparse cell, must have in its record the information needed to determine which segments of the subject trajectory are responsible for the occupancy of the cell.

4.1.2.3 Component Design Logic

The processing method of Buffer Cell Generator is given by the PDL in Figure 4-11. Its calling sequence is as follows.

```
Buffer Cell Generator
    Decode
    Encode
```

```
ROUTINE Buffer_Generator;
#Component of Cell Grid Chain Generator#
PARAMETERS T_Present IN, T_Horizon IN, Subject_Fl_Id IN;
REFER TO GLOBAL SPARSE_CELLS IN;
REFER TO SHARED LOCAL BUFFER_CELLS OUT;
DEFINE VARIABLES
   T_Present    Current time
   T_Horizon    Current time horizon
   Temp_Node    A temporary value for a node ID
   T_Cell       Time at center of sparse cell in quantized cell
                   coordinates
   Y_Cell       Y point at center of sparse cell in quantized cell
                   coordinates
   X_Cell       X point at center of sparse cell in quantized cell
                   coordinates
   T            Index for iteration on time
   Y            Index for iteration on horizontal y
   X            Index for iteration on horizontal x;
```

FIGURE 4-11
BUFFER_CELL_GENERATOR

```
    REPEAT FOR EACH SPARSE_CELLS RECORD
       WHERE SPARSE_CELLS.fl_id EQ Subject_Fl_Id;
       Temp_Node = SPARSE_CELLS.tree_node_id;
       CALL Decode (Temp_Node IN, T_Cell OUT, Y_Cell OUT, X_Cell
          OUT);
       FOR T = T_Cell - 1 TO T_Cell + 1;
          IF (T GT T_Present AND T LE T_Horizon)
          THEN
             FOR Y = Y_Cell - 1 TO Y_Cell + 1;
                FOR X = X_Cell - 1 TO X_Cell + 1;
                   CALL Encode (T IN, Y IN, X IN, Temp_Node OUT);
                   #Check to see if the buffer cell is already in#
                      #BUFFER_CELLS.  If it is, update its record.#
                      #Otherwise, create a record for it.#
                   IF (COUNT(BUFFER_CELLS.node_id EQ Temp_Node)) NE 0
                   THEN
                      UPDATE IN BUFFER_CELLS
                         (min_z = MIN(min_z, SPARSE_CELLS.min_z),
                         max_z = MAX(max_z, SPARSE_CELLS.max_z),
                         entry_time = MIN(entry_time, SPARSE
                         CELLS.entry_time), exit_time = MAX(exit
                         time, SPARSE_CELLS.exit_time))
                         WHERE BUFFER_CELLS.node_id EQ Temp_Node;
                   ELSE
                      INSERT INTO BUFFER_CELLS (node_id = Temp_Node,
                         min_z = SPARSE_CELLS.min_z, max_z = SPARSE_
                         CELLS.max_z, entry_time = SPARSE_CELLS.
                         entry_time, exit_time = SPARSE_CELLS.exit_
                         time);
END Buffer_Cell_Generator;
```

FIGURE 4-11
BUFFER_CELL_GENERATOR (Concluded)

Buffer Cell Generator searches through the records of SPARSE_CELLS selecting out any record associated with the current subject flight identifier. The (integer) cell coordinates are obtained using a utility routine called Decode, which performs the inverse operation to Encode discussed in Section 4.1.1.3. Then, each of the three cell dimensions are processed in nested loops to create buffer cells at all positions situated within 1 cell coordinate of the sparse cell. The nearest diagonal and orthogonal neighbors of the sparse cell are thus selected as buffer cells.

Then the various attributes of the sparse cell are transferred to the buffer cells. If the buffer cell is not represented in BUFFER_CELLS, the sparse cell attributes (minimum altitude, maximum altitude, entry time and exit time) are transferred directly to the buffer cell record along with the node identifier returned from Encode. If the buffer cell already exists, it is updated so that the buffer cell attributes have the greatest necessary range. In this manner, the buffer cell records, like the sparse cell records, provide vertical separation assurance and the time information required by the Coarse Filter.

4.1.3 Grid To Tree Converter

4.1.3.1 Mission

The Grid To Tree Converter component of the Grid Chain Generator uses applicable SPARSE_CELLS and BUFFER_CELLS records to build the subject's sparse and buffer trees.

4.1.3.2 Design Consideration and Component Environment

Before entering the Grid To Tree Converter component, the global SPARSE_CELLS table and the (local) BUFFER_CELLS table have been completed for the subject trajectory. Grid To Tree Converter takes the applicable node identifiers of the sparse and buffer cells and generates the necessary node-parent relationships to establish the tree structure. This facilitates the Coarse Filter's process of ruling out object trajectories that are well separated in space or time from the subject.

Input

The inputs to Grid To Tree Converter include the SPARSE_CELLS and BUFFER_CELLS tables and the subject flight identifier.

Output

The outputs are the shared local tables SPARSE_TREE and BUFFER_TREE, which contain information necessary for tree manipulations.

4.1.3.3 Component Design Logic

The PDL representation of the Grid to Tree Converter is given in Figure 4-12.

The Grid to Tree Converter component takes each record of the SPARSE_CELLS table applicable to the subject and creates the tree structure starting at the leaf level and working back to the root.

All of the information needed for determining parentage is contained in the cell identifier. This information is extracted using the utility Get Parent. Consider the example of a coordinate-to-node identifier mapping described in Section 4.1.1.3 and illustrated in Figure 4-7. For this mapping, Get Parent would drop the rightmost digit of a node identifier (say, node 022 in Figure 2-4) to yield the node identifier of its parent (02).

Results, consisting of temporary node identifiers and the flight identifier from the sparse cells, are incorporated in the SPARSE_TREE and the BUFFER_TREE tables.

4.2 Coarse Filter

The Coarse Filter component of Flight Plan Conflict Probe is designed to reduce the number of object aircraft that need to be analyzed by the Fine Filter for possible conflict with the subject aircraft. It accomplishes this task by identifying the aircraft that occupy the same general region in space and time as the subject aircraft and eliminating all others from further consideration. Specifically, the Coarse Filter invokes a recursive routine called Nominee Detection. Nominee Detection eliminates, at every stage of the recursion, all those object aircraft whose trajectories do not occupy any of the blocks that contain subject aircraft buffer cells. The algorithm invokes itself to check each of the eight sub-blocks for co-occupancy. It continues dividing co-occupied blocks and eliminating non-neighboring object aircraft until the cell level of the grid is reached. Thereupon, any object aircraft occupying a buffer cell of the subject aircraft is tested for an altitude separation violation with the subject aircraft for

```
ROUTINE Grid_To_Tree_Converter;
#Component of Grid Chain Generator#
PARAMETERS Subject_Fl_Id IN;
REFER TO GLOBAL SPARSE_CELLS IN, BUFFER_CELLS IN;
REFER TO SHARED LOCAL SPARSE_TREE OUT, BUFFER_TREE OUT, MAX_LEVEL IN;
DEFINE VARIABLES
   Temp_Node   Temporary storage of node_id
   Temp_Child  Temporary storage of child_id
   Level       Temporary tree level counter;
   REPEAT FOR EACH SPARSE_CELLS RECORD
      WHERE SPARSE_CELLS.fl_id EQ Subject_Fl_Id;
      Temp_Child = SPARSE_CELLS.tree_node_id;
      FOR Level = Max_Level TO 0;
         CALL Get_Parent (Temp_Child IN, Level IN, Temp_Node OUT);
         INSERT INTO SPARSE_TREE (fl_id = SPARSE_CELLS.fl_id,
            node_id = Temp_Node, child_id = Temp_Child);
         Temp_Child = Temp_Node;
   REPEAT FOR EACH BUFFER_CELLS RECORD;
      Temp_Node = BUFFER_CELLS.node_id;
      FOR Level = Max_Level TO 0;
         CALL Get_Parent (Temp_Child IN, Level IN, Temp_Node OUT);
         INSERT INTO BUFFER_TREE (node_id = Temp_Node,
            child_id = Temp_Child);
         Temp_Child = Temp_Node;
END Grid_To_Tree_Converter;
```

FIGURE 4-12
GRID_TO_TREE_CONVERTER

these cells. This test is performed by invoking the element Nominee Detection Altitude Test. Those object aircraft for which this test indicates a violation of the vertical separation criterion are identified as nominees. The nominees are passed on to the Fine Filter where a more thorough analysis involving the segments of the conflicting aircraft is conducted.

Figure 2-3 shows the planning region grid and the eight sub-blocks of each stage of this routine. As indicated in Section 2.1.8, octal tree data structures are used to represent the planning region blocks and cells occupied by the subject and object aircraft.

Figure 4-13 illustrates the Coarse Filter organizational hierarchy. Flight Plan Conflict Probe calls the Coarse Filter which, in turn, invokes Nominee Detection, its single component.

4.2.1 Nominee Detection

4.2.1.1 Mission

The purpose of this component of the Coarse Filter is to identify all cells within the planning region's three-dimensional (x,y,t) grid which are buffer cells of the subject aircraft and are occupied by at least one object aircraft. For each subject-object aircraft pair so identified, an altitude check is performed to determine if the altitude ranges (minimum to maximum) of the aircraft over the cell are such that the aircraft may violate the vertical separation criterion. This test is performed by invoking the element Nominee Detection Altitude Test. Those object aircraft that are found to violate this criterion with the subject aircraft are labeled nominees and are placed in the NOMINEES table and passed to the Fine Filter for further processing.

4.2.1.2 Design Considerations and Component Environment

Input

The information supplied to the Nominee Detection algorithm consists of parameters, global data, and shared local data. The parameters consist of Current_Node_Id, Level, Test_Time_Begin, and Test_Time_End. The Current_Node_Id is used in the recursion to identify the node being processed (in preorder) at a given level of the recursion. The parameter Level identifies which level of recursion is currently being processed. Test_Time_Begin and Test_Time_End are used to limit the scope of the search. The algorithm will search only those blocks

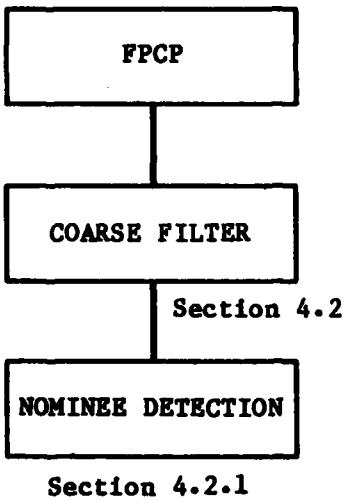


FIGURE 4-13
COARSE FILTER ORGANIZATIONAL STRUCTURE

whose time interval overlaps with these test times. For example, if the Coarse Filter is called because of a horizon update with Test_Time_Begin equal to the previous time horizon and Test_Time_End equal to the new time horizon, there is no need to check blocks associated with times outside of this interval.

The shared local tables and variables input into the algorithm are the ALLOBJECT_TREE, BUFFER_TREE, BUFFER_CELLS, Max_Level, Real_Subject_Fl_.d, and Trial_Flag. ALLOBJECT_TREE and BUFFER_TREE are tables which define the structure of the trees. These tables contain a record for each parent-child relationship in the tree. When the preorder traversal reaches a record including a leaf node (always as a child), the leaf (child) identifier is used to reference the corresponding cell data in BUFFER_CELLS. This information is used to provide the information on the subject aircraft segment in that cell to the NOMINEES table. The global table SPARSE_CELLS provides the corresponding data for the leaf level of the ALLOBJECT_TREE. Max_Level defines the maximum level of division used to reach the leaf level. The variable Trial_Flag is set to TRUE if the call to Nominee Detection is due to a trial probe and to FALSE otherwise. In case of a trial probe, the variable Real_Subject_Fl_Id contains the flight identifier of the flight being probed. This is used to avoid detecting potential conflicts of the trial trajectory and the actual trajectory of the flight being trial probed. (Real_Subject_Flight_Id is not used for a trajectory update or a horizon update.) Finally the global variable Current_Time is used to limit the search to cells that correspond to those portions of trajectories that occur in the future.

Output

The output of the Nominee Detection algorithm is the shared local NOMINEES table which contains the following information for each subject buffer cell occupied by an object (nominee):

1. the flight identifier of the nominee aircraft

2. the node identifier of the airspace cell where the co-occupancy occurs

3. the subject entry time and subject exit time, which specify the t coordinates of two cusps on the subject's trajectory: a) the earlier cusp on the earliest segment which causes the cell to be declared occupied by the subject and b) the later cusp on the

latest segment which causes the cell to be declared occupied by the subject

4. the nominee entry time and nominee exit time, which specify the t coordinates of two cusps along the nominee's trajectory, defined as in 3a and 3b above

4.2.1.3 Component Design Logic

Nominee Detection is a recursive algorithm which uses preordering to traverse (in parallel) those branches which occur in both BUFFER_TREE (which reflects the trajectory of the subject aircraft) and ALLOBJECT_TREE (which reflects the trajectories of all of the objects). A description of tree traversal methods is given in Appendix C. Each call to Nominee Detection considers a node that occurs in both trees, which is input as Current_Node_Id. On the first call to Nominee Detection, this node is the root of both trees. The Current_Node_Id is used as a key to locate records corresponding to its children in both trees. Only children common to both trees are considered.

Two time tests are performed. First a check to see if the time intervals associated with the current child block exceeds Current_Time. If so, testing for this child ends, since the aircraft has already passed this point in time. Next, the Test_Time_Begin and Test_Time_End are tested against the given block's time intervals to see if they overlap.

For each pair of children satisfying the above conditions, Nominee Detection calls itself with the child_id of the subject's tree as the new Current_Node_Id. At every stage of the recursion, the algorithm also checks to see if the cell level (i.e., Max_Level) has been reached. If it has been reached, then the algorithm locates and retrieves the segment data for the subject in BUFFER_CELLS. Included in this data are the minimum and maximum altitudes attained for the segment. Next, the algorithm locates, iteratively, each SPARSE_CELLS record of every object aircraft co-occupied with the subject in the cell and retrieves the corresponding segment data for that aircraft. For the current object aircraft, a comparison is made of its altitude range in the cell with that of the subject aircraft. If the vertical separation between the aircraft is less than that required by the vertical separation criterion, then the object aircraft is labeled a nominee and the NOMINEES table is updated with the appropriate data. The test for vertical separation is performed by the

Nominee Detection Altitude Check element. Figure 4-14 gives the PDL for the Nominee Detection component.

Nominee Detection Altitude Check

The Nominee Detection Altitude Check element performs a test to determine whether or not the subject and object (candidate for a nominee) segments violate the vertical separation criterion Vert_Sep. The algorithm first determines whether or not the maximum altitude attained by either aircraft segment is larger than 29,000 feet. If so, the vertical separation criterion Vert_Sep is set equal to the global parameter Sepz_Hi; otherwise, it is set equal to Sepz_Lo.

Finally, the algorithm determines if the vertical distance between the segments is within Vert_Sep. If it is, the status is set to a "nominee" status; otherwise, it is set to a "no nominee" status.

Figure 4-15 is a PDL representation of the Nominee Detection Altitude Check element.

4.3 Fine Filter

The Fine Filter component of Flight Plan Conflict Probe identifies the encounters of a particular subject aircraft from the data in the aircraft's NOMINEES Table. Unlike the Coarse Filter, which is essentially a search and copy subfunction, the Fine Filter involves detailed mathematical analyses conducted on aircraft segments. (The word "segment" in the following paragraphs will be used to denote either a regular segment or a cusp pair representing entry and exit points of a holding pattern or vertical maneuver.)

The first task of the Fine Filter algorithm is to create a list of subject-nominee segment pairs consisting of those subject and nominee segments which are associated with a co-occupied cell identified in NOMINEES. Each such pair is compared to those segment pairs previously analyzed by the Fine Filter for possible repetition. The objective of this test is to assure that no segment pair undergoes the same mathematical analysis more than once. If the segment pair has not yet been processed, the segments are tested for possible overlap in time by the Time Check component. Those segments that overlap are checked by the Altitude Check component for possible violation of the vertical separation criterion. If violation is detected, the segments are tested by the Horizontal Check component, within their common time interval, to determine

```
ROUTINE Nominee_Detection;
PARAMETERS Current_Node_Id IN, Level IN, Test_Time_Begin IN, Test_
   Time_End IN;
REFER TO SHARED LOCAL ALLOBJECT_TREE IN, BUFFER_TREE IN, BUFFER_
   CELLS IN, NOMINEES OUT, Max_Level IN, Real_Subject_Fl_Id IN,
   Trial_Flag IN;
REFER TO GLOBAL SPARSE_CELLS IN, CURRENT_TIME IN;
DEFINE VARIABLES
```

Current_Node_Id	Identifier of the current root of the subtree being traversed
Level	Current level of the root of the subtree in the parent tree
Test_Time_Begin	Time at which testing for co-occupancy begins
Test_Time_End	Time at which testing for co-occupancy ends
Subject_Min_Z	The lowest altitude through which this flight plan trajectory passes in this cell
Subject_Max_Z	The highest altitude through which this flight plan trajectory passes in this cell
Subject_Entry_Time	Cusp which precedes entry into this cell
Subject_Exit_Time	Cusp which follows exit from this cell
Status	An indicator used to specify whether a given object aircraft is or is not a nominee;

FIGURE 4-14
NOMINEE_DETECTION

```
IF Level LT Max_Level
THEN
   REPEAT FOR EACH BUFFER_TREE RECORD
      #for each child of a node, a record exists with node_id as#
         #first field and child_id as second field#
      WHERE BUFFER_TREE.node_id EQ Current_Node_Id;
      IF the time interval associated with the Current_Node_Id
         is in the interval (Test_Time_Begin,Test_Time_End)
         AND CURRENT_TIME.time does not exceed this time interval
      THEN #check for matching child block in the ALLOBJECT tree#
         #if match then blocks are co-occupied#
         IF COUNT(ALLOBJECT_TREE.node_id EQ Current_Node_Id AND
         ALLOBJECT_TREE.child_id EQ BUFFER_TREE.child_id) NE 0
         THEN #child was found, check next level in pre-order,#
            #to find which cells are co-occupied with the subject#
            CALL Nominee_Detection(BUFFER_TREE.child_id IN, Level+1
               IN, Test_Time_Begin IN, Test_Time_End IN);
ELSE #at the leaf level (cell level) of both trees,#
   #determine all objects in the cell to be considered nominees;#
      #get altitudes and segments in this cell for the subject#
   SELECT FIELDS min_z, max_z, entry_time, exit_time
      INTO Subject_Min_Z, Subject_Max_Z, Subject_Entry_Time,
         Subject_Exit_Time
      FROM BUFFER_CELLS
      WHERE BUFFER_CELLS.tree_node_id EQ Current_Node_Id;
   REPEAT FOR EACH SPARSE_CELLS RECORD #for each co-occupied object#
      WHERE SPARSE_CELLS.tree_node_id EQ Current_Node_Id;
      #if this is a trial probe, ignore the real subject#
      IF ((Trial_Flag EQ FALSE) OR (Trial_Flag EQ TRUE AND
         SPARSE_CELLS.Fl_Id NE Real_Subject_Fl_Id))
      THEN #test for altitude violation between aircraft#
         CALL Nominee_Detection_Altitude_Check (Subject_Min_Z,
            Subject_Max_Z, SPARSE_CELLS.min_z, SPARSE_CELLS.max_z,
            Status);
         IF Status = 'nominee'
         THEN #this object is nominee, add segments for the subject#
            #and this object to the Nominee table#
            INSERT INTO NOMINEES
               (Fl_id = SPARSE_CELLS.Fl_id, node_id = SPARSE_
               CELLS.tree_node_id, subject_entry_time = Subject_
               Entry_Time, subject_exit_time = Subject_Exit_Time,
               nominee_entry_time = SPARSE_CELLS.entry_time,
               nominee_exit_time = SPARSE_CELLS.exit_time);
END Nominee_Detection;
```

FIGURE 4-14
NOMINEE DETECTION (Concluded)

```
ROUTINE Nominee_Detection_Altitude_Check;
#this routine determines if the required vertical separation#
   #distance is maintained between two aircraft in a given cell#
PARAMETERS Subject_Min_Z IN, Subject_Max_Z IN, Object_Min_Z IN,
   Object_Max_Z IN, Status OUT;
REFER TO GLOBAL Sepz_Hi IN, Sepz_Lo IN;
DEFINE VARIABLES
   Vert_Sep        Vertical separation criterion
   Subject_Min_Z   The lowest altitude through which subject's flight
                      plan trajectory passes in this cell
   Subject_Max_Z   The maximum altitude through which subject's
                      flight plan trajectory passes in this cell
   Object_Min_Z    The lowest altitude through which object's flight
                      plan trajectory passes in this cell
   Object_Max_Z    The maximum altitude through which object's flight
                      plan trajectory passes in this cell
   Status          An indicator used to specify whether a given object
                      aircraft is or is not a nominee;
   #determine required vertical separation#
   #If the maximum altitude of either aircraft exceeds the altitude#
      #at which the minimum separation requirement changes, set the#
      #maximum distance#
   IF MAX (Subject_Min_Z, Subject_Max_Z, Object_Min_Z, Object_Max_Z)
      GT 29000 feet
   THEN
      Vert_Sep = Sepz_Hi;
   ELSE
      Vert_Sep = Sepz_Lo;
   #determine if the vertical distance between segments in the cell#
   #are in violation#
   IF (MAX (Subject_Min_Z, Object_Min_Z) - MIN (Subject_Max_Z,
      Object_Min_Z)) LT Vert_Sep
   THEN
      Status = 'nominee';
   ELSE
      Status = 'no nominee';
END Nominee_Detection_Altitude_Check;
```

FIGURE 4-15
NOMINEE_DETECTION_ALTITUDE_TEST

whether or not the horizontal separation of the aircraft is less than the advisory separation criterion (Advisory Seph) and, if so, whether or not it is also less than the priority separation criterion (Priority Seph). The component distinguishes between regular segments and segments associated with a holding pattern or vertical maneuver in its calculation of the distance separating the two aircraft. If a violation of the advisory separation criterion is detected, an entry is made in the ENCOUNTERS Table by the Encounter List Builder describing details of the event. This table includes encounter data for all of the aircraft in the planning region.

Once all segment pairs associated with a co-occupied cell have been tested, the Fine Filter repeats the entire process for the next co-occupied cell, etc., until all cells are processed.

Figure 4-16 illustrates the Fine Filter organizational hierarchy. Flight Plan Conflict Probe invokes the Fine Filter after the Coarse Filter, whereupon the Fine Filter calls its various components in tandem.

Figure 4-17 contains a glossary of the local variables and tables which are common to at least two components of the Fine Filter. It should be used as a supplement to the PDL descriptions of the various algorithms provided in the figures of Section 4.3.

Figure 4-18 is a PDL presentation of the Fine Filter component.

4.3.1 Segment Pair Builder

4.3.1.1 Mission

The mission of the Segment Pair Builder is to organize the subject aircraft and nominee aircraft cusps in the global data base TRAJECTORIES into subject-nominee segment pairs that are associated with each co-occupied cell identified by the Coarse Filter.

4.3.1.2 Design Considerations and Component Environment

Input

The inputs to the Segment Pair Builder Component include the unique identifiers for the subject and nominee aircraft trajectories, Subject_Fl_Id and Nominee_Fl_Id, respectively, and the times of the first and last subject aircraft and nominee aircraft cusps associated with a co-occupied cell. These times

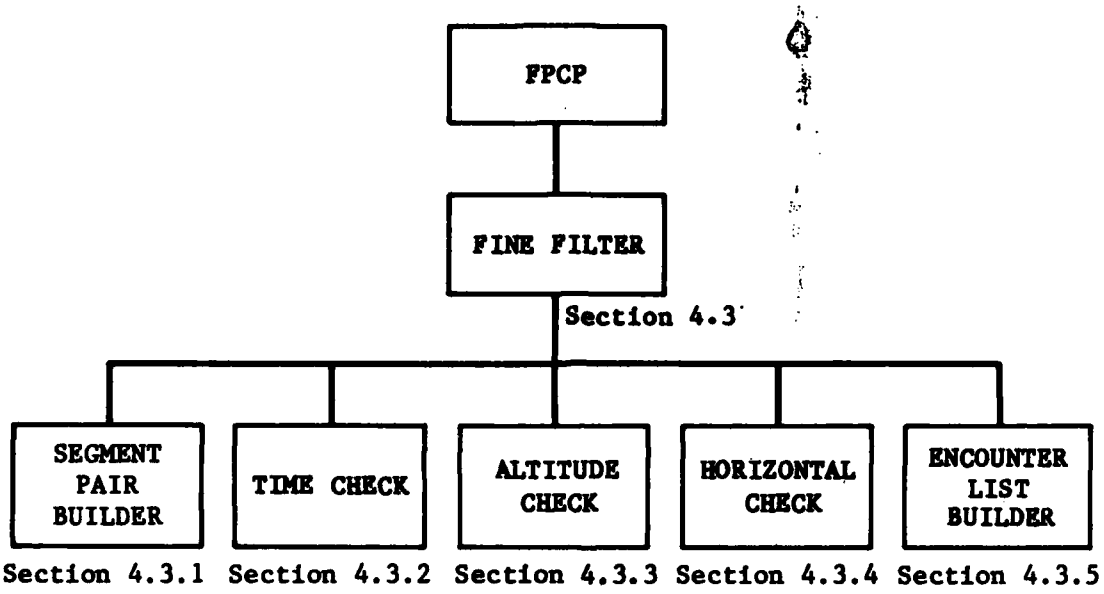


FIGURE 4-16
FINE FILTER ORGANIZATIONAL STRUCTURE

VARIABLES

Advisory_Time_Viol_End	Latest time that the advisory horizontal separation criterion is violated
Advisory_Time_Viol_Start	Earliest time that the advisory horizontal separation criterion is violated
Msep_Dist	Minimum separation distance between the aircraft in the horizontal plane
Nominee_Fl_Id	Unique identifier for the flight plan of the nominee aircraft
Nominee_Viol_End_Pt	Spatial coordinates of the nominee aircraft at the end of the advisory violation
x	X coordinate
y	Y coordinate
z	Z coordinate
Nominee_Viol_Start_Pt	Spatial coordinates of the nominee aircraft at the start of the advisory violation
x	X coordinate
y	Y coordinate
z	Z coordinate
Priority_Time_Viol_End	Latest time that the priority horizontal separation criterion is violated
Priority_Time_Viol_Start	Earliest time that the priority horizontal separation criterion is violated
Subject_Fl_Id	Unique identifier for the flight plan of the subject aircraft
Subject_Viol_End_Pt	Spatial coordinates of the subject aircraft at the end of the advisory violation
x	X coordinate
y	Y coordinate
z	Z coordinate

FIGURE 4-17
FINE FILTER GLOSSARY

Subject_Viol_Start_Pt	Spatial coordinates of the subject aircraft at the start of the advisory violation
x	X coordinate
y	Y coordinate
z	Z coordinate
Time_Msep	Time of minimum separation between the aircraft in the horizontal plane
Time_Overlap_Max	Latest time that the subject and nominee segments overlap in time
Time_Overlap_Min	Earliest time that the subject and nominee segments overlap in time

TABLES

NOMINEE_SEGMENT	Pair of cusps representing the nominee segment being processed
first_t	Time of the first cusp
first_x	X coordinate of the first cusp
first_y	Y coordinate of the first cusp
first_z	Z coordinate of the first cusp
first_cusp_type	Cusp type of the first cusp
second_t	Time of the second cusp
second_x	X coordinate of the second cusp
second_y	Y coordinate of the second cusp
second_z	Z coordinate of the second cusp
second_cusp_type	Cusp type of the second cusp
first_point AGGREGATE (first_x, first_y, first_z)	
second_point AGGREGATE (second_x, second_y, second_z)	
first_xy_pair AGGREGATE (first_x, first_y)	
second_xy_pair AGGREGATE (second_x, second_y)	
hz_first_vtx AGGREGATE (first_x,first_y)	
hz_sec_vtx AGGREGATE (second_x,second_y)	

FIGURE 4-17
FINE FILTER GLOSSARY (Continued)

```
SEGMENT_PAIR_LIST              Table containing the subject and
                                  nominee segment pairs for a
                                  co-occupied cell identified by the
                                  Coarse Filter
   subj_first_t                Time of subject's first cusp
   subj_first_x                X coordinate of subject's first cusp
   subj_first_y                Y coordinate of subject's first cusp
   subj_first_z                Z coordinate of subject's first cusp
   subj_first_cusp_type        Cusp type of subject's first cusp
   subj_second_t               Time of subject's second cusp
   subj_second_x               X coordinate of subject's second
                                  cusp
   subj_second_y               Y coordinate of subject's second
                                  cusp
   subj_second_z               Z coordinate of subject's second
                                  cusp
   nominee_first_t             Time of nominee's first cusp
   nominee_first_x             X coordinate of nominee's first cusp
   nominee_first_y             Y coordinate of nominee's first cusp
   nominee_first_z             Z coordinate of nominee's first cusp
   nominee_first_cusp_         Cusp type of nominee's first cusp
      type
   nominee_second_t            Time of nominee's second cusp
   nominee_second_x            X coordinate of nominee's second
                                  cusp
   nominee_second_y            Y coordinate of nominee's second
                                  cusp
   nominee_second_z            Z coordinate of nominee's second
                                  cusp
   nominee_second_cusp_        Cusp type of nominee's second cusp
      type
   subject_segment AGGREGATE (** first 10 fields **)
   nominee_segment AGGREGATE (** last 10 fields **)
   subject_nominee_segment_pair AGGREGATE (** all 20 fields **)
SUBJECT_SEGMENT                Pair of cusps representing the
                                  subject segment being processed;
                                  fields defined like NOMINEE
                                  SEGMENT above
```

FIGURE 4-17
FINE FILTER GLOSSARY (Concluded)

```
ROUTINE Fine_Filter;
PARAMETERS Subject_Fl_Id IN;
REFER TO GLOBAL ENCOUNTERS INOUT, PRIOR_ENCOUNTERS OUT;
REFER TO SHARED LOCAL NOMINEES IN;
DEFINED IN GLOSSARY
   Subject_Fl_Id
   Time_Overlap_Min
   Time_Overlap_Max
   Advisory_Time_Viol_Start
   Advisory_Time_Viol_End
   Priority_Time_Viol_Start
   Priority_Time_Viol_End
   Time_Msep
   Msep_Dist
   SEGMENT_PAIR_LIST;
DEFINE VARIABLES
   Status                    Variable indicating the outcome of a
                                particular Fine Filter test;
DEFINE TABLES
   PROCESSED_SEGMENT_        Subject-nominee segment pairs
      PAIR_LIST                 previously processed by the Fine
                                Filter; fields defined like
                                SEGMENT_PAIR_LIST in Glossary;
```

FIGURE 4-18
FINE FILTER

```
# produce copy of ENCOUNTERS Table for Sector Workload Probe #
PRIOR_ENCOUNTERS = ENCOUNTERS;
# repeat for each co-occupied cell identified by the Coarse Filter #
REPEAT FOR EACH NOMINEES RECORD;
   # organize cusps into subject-nominee segment pairs #
   CALL Segment_Pair_Builder (Subject_Fl_Id IN, NOMINEES.fl_id IN,
      NOMINEES.subject_entry_time IN, NOMINEES.subject_exit_time IN,
      NOMINEES.nominee_entry_time IN, NOMINEES.nominee_exit_time IN,
      SEGMENT_PAIR_LIST OUT);
   REPEAT FOR EACH SEGMENT_PAIR_LIST RECORD;
      WHERE SEGMENT_PAIR_LIST.subject_nominee_segment_pair IS NOT IN
         PROCESSED_SEGMENT_PAIR_LIST; # repeat only for subject- #
         # nominee segment pairs not previously processed        #
      INSERT INTO PROCESSED_SEGMENT_PAIR_LIST
         (subject_nominee_segment_pair = SEGMENT_PAIR_LIST.subject_
         nominee_segment_pair);
      # conduct tests of the subject-nominee segment pair in the #
      # various dimensions                                       #
      CALL Time_Check (SEGMENT_PAIR_LIST.subject_segment IN,
         SEGMENT_PAIR_LIST.nominee_segment IN, Status OUT, Time_
         Overlap_Min OUT, Time_Overlap_Max OUT);
      IF Status EQ 'Time intervals overlap'
      THEN
         CALL Altitude_Check (SEGMENT_PAIR_LIST.subject_segment IN,
            SEGMENT_PAIR_LIST.nominee_segment IN, Status OUT);
         IF Status EQ 'Violation of vertical separation criterion'
         THEN
            CALL Horizontal_Check (SEGMENT_PAIR_LIST.subject_segment
               IN, SEGMENT_PAIR_LIST.nominee_segment IN, Subject_Fl_
               Id IN, NOMINEES.fl_id IN, Time_Overlap_Min IN, Time_
               Overlap_Max IN, Status OUT, Advisory_Time_Viol_Start
               OUT, Advisory_Time_Viol_End OUT, Priority_Time_Viol_
               Start OUT, Priority_Time_Viol_End OUT, Time_Msep OUT,
               Msep_Dist OUT);
            IF Status EQ 'Violation of advisory horizontal separation
               criterion'
            THEN   # store the encounter data in a table #
               CALL Encounter_List_Builder (SEGMENT_PAIR_LIST.
                  subject_segment IN,SEGMENT_PAIR_LIST.nominee_
                  segment IN, Subject_Fl_Id IN, NOMINEES.fl_id IN,
                  Advisory_Time_Viol_Start IN, Advisory_Time_Viol_
                  End IN, Priority_Time_Viol_Start IN, Priority_Time_
                  Viol_End IN, Time_Msep IN, Msep_Dist IN);
END Fine_Filter;
```

FIGURE 4-18
FINE FILTER (Concluded)

are denoted by the local variables Subject_Entry_Time, Subject_Exit_Time, Nominee_Entry_Time, and Nominee_Exit_Time. In addition to all of these variables which are provided by the Fine Filter routine, the Segment Pair Builder Component uses the global table TRAJECTORIES from which it obtains the cusp information for the two aircraft.

Output

The output of the Segment Pair Builder Component is a table, SEGMENT_PAIR_LIST, which contains in each record a pair of segments, one from each aircraft, that are associated with a specific co-occupied cell identified by the Coarse Filter. Each segment in a record is a pair of cusps. The subject segment and the nominee segment in each record are passed on to other components of the Fine Filter for specific testing.

4.3.1.3 Component Design Logic

The Segment Pair Builder component is called by the Fine Filter for every co-occupied cell in the NOMINEES Table. Using the parameters Subject_Entry_Time and Subject_Exit_Time (obtained from this table) to identify the subject cusps associated with the cell, it selects these cusps from the TRAJECTORIES Table and sorts them in increasing order of time. Every consecutive pair of cusps in the resulting list represents a segment in the trajectory of the aircraft. The component then repeats the process for the nominee cusps associated with the same cell. Finally, it combines subject and nominee segments from each list into pairs, storing every possible combination in a local table called SEGMENT_PAIR_LIST.

Figure 4-19 shows the PDL representation of the Segment Pair Builder algorithm.

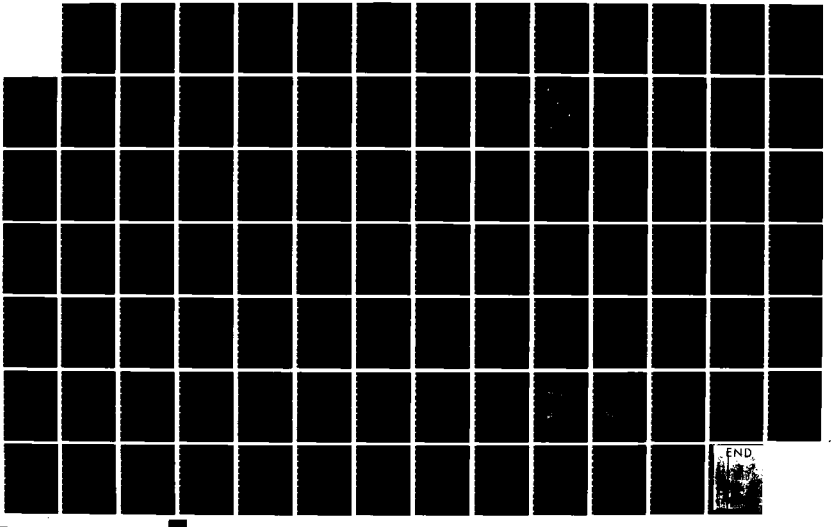
4.3.2 Time Check

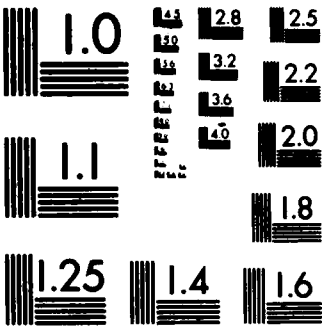
4.3.2.1 Mission

The mission of the Time Check component of the Fine Filter is simply to determine if the pair of segments under consideration overlap in time and, if so, to calculate the endpoints of the common time interval.

AD-A136 795 AUTOMATED EN ROUTE AIR TRAFFIC CONTROL ALGORITHMIC SPECIFICATIONS VOLUME 3.. (U) FEDERAL AVIATION ADMINISTRATION WASHINGTON DC SYSTEMS ENGINEE.. 2/2

UNCLASSIFIED W P NIEDRINGHAUS ET AL. SEP 83 F/G 17/7 NL





```
ROUTINE Segment_Pair_Builder;
PARAMETERS Subject_Fl_Id IN, Nominee_Fl_Id IN, Subject_Entry_Time IN,
   Subject_Exit_Time IN, Nominee_Entry_Time IN, Nominee_Exit_Time IN,
   SEGMENT_PAIR_LIST OUT;
REFER TO GLOBAL TRAJECTORIES IN;
DEFINED IN GLOSSARY
   Subject_Fl_Id
   Nominee_Fl_Id
   SEGMENT_PAIR_LIST;
DEFINE VARIABLES
   Ordered_Subject_
      Cusps (*,5)          Array whose rows are subject cusps
                              ordered according to increasing time
   Subject_Entry_Time      Time of the first subject aircraft cusp
                              associated with the co-occupied cell
   Subject_Exit_Time       Time of the last subject aircraft cusp
                              associated with the co-occupied cell
   Subject_Cusp_Count      Total number of rows in Ordered_
                              Subject_Cusps
   Ordered_Nominee_
      Cusps (*,5)          Array whose rows are nominee cusps
                              ordered according to increasing time
   Nominee_Entry_Time      Time of the first nominee aircraft cusp
                              associated with the co-occupied cell
   Nominee_Exit_Time       Time of the last nominee aircraft cusp
                              associated with the co-occupied cell
   Nominee_Cusp_Count      Total number of rows in Ordered_
                              Nominee_Cusps
   I                       Row index for Ordered_Subject_Cusps
   J                       Row index for Ordered_Nominee_Cusps;
```

FIGURE 4-19
SEGMENT PAIR BUILDER

```
# select the subject cusps that are associated with the co-occupied #
# cell and order them by increasing time                            #
SELECT FIELDS cusp, cusp_type
   FROM TRAJECTORIES
   INTO Ordered_Subject_Cusps
   WHERE (TRAJECTORIES.fl_id EQ Subject_Fl_Id) AND
      (TRAJECTORIES.time GE Subject_Entry_Time AND
      TRAJECTORIES.time LE Subject_Exit_Time)
   ORDERED BY TRAJECTORIES.time
   RETURN COUNT (Subject_Cusp_Count);

# select the nominee cusps that are associated with the co-occupied #
# cell and order them by increasing time                            #
SELECT FIELDS cusp, cusp_type
   FROM TRAJECTORIES
   INTO Ordered_Nominee_Cusps
   WHERE (TRAJECTORIES.fl_id EQ Nominee_Fl_Id) AND
      (TRAJECTORIES.time GE Nominee_Entry_Time AND
      TRAJECTORIES.time LE Nominee_Exit_Time)
   ORDERED BY TRAJECTORIES.time
   RETURN COUNT (Nominee_Cusp_Count);
# construct a table of subject-nominee segment pairs associated     #
# with the co-occupied cell                                         #
FOR I = 2 TO Subject_Cusp_Count;
   FOR J = 2 TO Nominee_Cusp_Count;
      INSERT INTO SEGMENT_PAIR_LIST
         (subject_segment = Ordered_Subject_Cusps(I-1,*) CONCAT
         Ordered_Subject_Cusps(I,*), nominee_segment = Ordered
         Nominee_Cusps(J-1,*) CONCAT Ordered_Nominee_Cusps(J,*));
END Segment_Pair_Builder;
```

FIGURE 4-19
SEGMENT PAIR BUILDER (Concluded)

4.3.2.2 Design Considerations and Component Environment

Input

The inputs to the Time Check component are two tables, the SUBJECT_SEGMENT and the NOMINEE_SEGMENT, which are obtained from the SEGMENT_PAIR_LIST created by the Segment Pair Builder Component. The tables contain data describing the subject and nominee segments under consideration.

Output

The output of this component consists of a variable called Status which indicates whether or not the time intervals of the two segments overlap and two parameters, Time_Overlap_Min and Time_Overlap_Max, which are the bounds of the common interval, if such an interval exists.

4.3.2.3 Component Design Logic

The Time Check Component is called by the Fine Filter for each subject-nominee segment pair not previously processed by the Fine Filter. It establishes whether or not the two segments under consideration overlap in time simply by determining if either the subject aircraft segment occurs entirely before the nominee segment or, conversely, the nominee segment occurs entirely before the subject aircraft segment. If either case is true, Status is assigned a message indicating that the time intervals do not overlap. Otherwise, it is assigned a message designating that an overlap exists. The beginning and end of the overlap is computed by finding the latest starting time and earliest ending time, respectively, of the two segments.

Figure 4-20 is a PDL presentation of the Time Check Algorithm.

4.3.3 Altitude Check

4.3.3.1 Mission

The purpose of the Altitude Check Component of the Fine Filter is to analyze those segment pairs that pass through the Time Check for possible violations of the vertical separation criterion by their respective aircraft.

4.3.3.2 Design Considerations and Component Environment

If either aircraft is transitioning in altitude over its segment (this may include an aircraft in a holding pattern with

```
ROUTINE Time_Check;
PARAMETERS SUBJECT_SEGMENT IN, NOMINEE_SEGMENT IN, Status OUT,
   Time_Overlap_Min OUT, Time_Overlap_Max OUT;
DEFINED IN GLOSSARY
   Time_Overlap_Min
   Time_Overlap_Max
   SUBJECT_SEGMENT
   NOMINEE_SEGMENT;
DEFINE VARIABLES
   Status          Variable indicating whether or not the subject and
                     nominee segments overlap in time;
# check to determine if subject segment ends before nominee segment #
# begins or nominee segment ends before subject segment begins      #
IF SUBJECT_SEGMENT.second_t LE NOMINEE_SEGMENT.first_t  OR
   NOMINEE_SEGMENT.second_t LE SUBJECT_SEGMENT.first_t
THEN
   Status = 'Time intervals do not overlap';
ELSE
   Status = 'Time intervals overlap';
   # calculate the endpoints of the overlap in time #
   Time_Overlap_Min = MAX (SUBJECT_SEGMENT.first_t,
      NOMINEE_SEGMENT.first_t);
   Time_Overlap_Max = MIN (SUBJECT_SEGMENT.second_t,
      NOMINEE_SEGMENT.second_t);
END Time_Check;
```

FIGURE 4-20
TIME CHECK

vertical extent), the subject-nominee segment pair is automatically considered a candidate for the Horizontal Check Component without further testing by the Altitude Check Component. Detailed analysis of such segment pairs by this component is considered infeasible given the relatively small number of cases expected to be eliminated from such an analysis. For all practical purposes, the Coarse Filter prefiltering of transitioning aircraft in the vertical dimension seems sufficient.

Input

The data to be input into the Altitude Check Component consist of the subject aircraft segment and the nominee segment in the form of the local tables SUBJECT_SEGMENT and NOMINEE_SEGMENT, and the vertical separation criteria, Sepz_Hi and Sepz_Lo. The separation criteria are global parameters which are the FPCP standards for aircraft flying above and below 29,000 feet, respectively.

Output

The output from the component is the variable Status which indicates whether or not the subject-nominee segment pair violates the vertical separation criterion.

4.3.3.3 Component Design Logic

The Altitude Check Component is invoked by the Fine Filter whenever a subject-nominee segment pair passes through the Time Check Component. A PDL representation of the Altitude Check Algorithm is presented in Figure 4-21. The segments are first checked for transitions in altitude. If at least one of the aircraft changes altitude over its segment, Status is assigned a message indicating that there is a violation of the vertical separation criterion and the algorithm is terminated. This allows the segment pair to proceed directly to the Horizontal Check for reasons indicated in Section 4.3.3.2.

If the flights are both level, the algorithm selects the appropriate value of the vertical separation criterion, Vert_Sep, on the basis of the altitudes of the two aircraft. Whenever the altitude of either aircraft exceeds 29,000 feet, Vert_Sep is set equal to Sepz_Hi. Otherwise, it is set equal to Sepz_Lo. The difference between Sepz_Hi and Sepz_Lo reflects the greater separation required at higher altitudes where the aircraft travel at faster speeds.

```
ROUTINE Altitude_Check;
PARAMETERS  SUBJECT_SEGMENT IN, NOMINEE_SEGMENT IN, Status OUT;
REFER TO GLOBAL Sepz_Hi IN, Sepz_Lo IN;
DEFINED IN GLOSSARY
   SUBJECT_SEGMENT
   NOMINEE_SEGMENT;
DEFINE VARIABLES
   Status          Variable indicating whether or not the subject and
                      nominee aircraft violate the vertical separation
                      criterion
   Vert_Sep        Vertical separation criterion;

# if either segment is associated with a vertical maneuver or a  #
# vertical hold, no altitude check is performed                  #
IF SUBJECT_SEGMENT.first_cusp_type EQ 'vertical maneuver' OR
   SUBJECT_SEGMENT.first_cusp_type EQ 'vertical hold' OR
   NOMINEE_SEGMENT.first_cusp_type EQ 'vertical maneuver' OR
   NOMINEE_SEGMENT.first_cusp_type EQ 'vertical hold'
THEN
   Status = 'Violation of vertical separation criterion';
ELSE
   # set value of vertical separation criterion according to #
   # whether or not either aircraft is above 29000 feet      #
   IF MAX (SUBJECT_SEGMENT.first_z, NOMINEE_SEGMENT.first_z)
      GT 29000
   THEN
      Vert_Sep = Sepz_Hi;
   ELSE
      Vert_Sep = Sepz_Lo;
   # determine whether or not vertical separation criterion is #
   # violated                                                  #
   IF ABS (SUBJECT_SEGMENT.first_z - NOMINEE_SEGMENT.first_z)
      GE Vert_Sep
   THEN
      Status = 'No violation of vertical separation criterion';
   ELSE
      Status = 'Violation of vertical separation criterion';
END Altitude_Check;
```

FIGURE 4-21
ALTITUDE CHECK

If the algorithm determines that the vertical distance between the two aircraft equals or exceeds Vert_Sep, it assigns to Status a message indicating that there is no altitude violation. Otherwise, Status is assigned a message indicating the occurrence of an altitude violation.

4.3.4 Horizontal Check

4.3.4.1 Mission

The Horizontal Check Component of the Fine Filter is designed to test those subject-nominee segments that filter through the Time Check and Altitude Check Components for possible violations of the FPCP advisory and priority horizontal separation criteria.

4.3.4.2 Design Considerations and Component Environment

The component distinguishes between regular segments and segments which correspond to a holding pattern or a vertical maneuver in its determination of whether or not the horizontal separation criteria are violated. The reason for the distinction between the two cases is that the procedures required to determine whether a violation occurs are fundamentally different for the two cases. In addition, the type of data that describes a violation for two regular segments differs from that which describes the violation involving at least one holding pattern or vertical maneuver segment.

Input

The Horizontal Check component requires, as local inputs, the subject aircraft segment (SUBJECT_SEGMENT), nominee aircraft segment (NOMINEE_SEGMENT), unique identifiers for the subject and nominee flight plans (Subject_Fl_Id and Nominee_Fl_Id), and the bounds on the time interval in common between the two aircraft (Time_Overlap_Min and Time_Overlap_Max). These inputs are provided by the Fine Filter. In addition to the local inputs, a global table called MANEUVER_ENVELOPES is used by one of the elements of the Horizontal Check Component to access information regarding any vertical maneuvers or holding patterns that may be associated with at least one of the aircraft.

Output

The output of the Horizontal Check component is Status, a variable indicating whether or not the advisory horizontal separation criterion is violated and, if so, a set of parameters describing the violation. If the priority horizontal separation criterion is also violated, then a set of parameters describing this violation are included in the output. The complete list of parameters include the Advisory_Time_Viol_Start, Advisory_Time_Viol_End, Priority_Time_Viol_Start, Priority_Time_Viol_End, the time of minimum separation in the horizontal plane, Time_Msep, and the minimum separation distance in the plane, Msep_Dist.

4.3.4.3 Component Design Logic

The Horizontal Check component is invoked by the Fine Filter whenever a subject-nominee segment pair passes through the Time Check and Altitude Check. It, in turn, calls one of two possible elements, the Regular Segment Horizontal Check or the Maneuver Envelope Horizontal Check, depending on whether or not both segments are regular or at least one of the segments is associated with a holding pattern or a vertical maneuver. Each of the elements invokes other routines, as indicated in the following representation of the organizational structure of the Horizontal Check Component and in paragraphs below.

```
Horizontal_Check
   Regular_Segment_Horizontal_Check
      Relative_Vectors
      Violation_Times
   Maneuver_Envelope_Horizontal_Check
      Maneuver_Envelope_Test
         Envelope_Envelope_Violation_Check
            Get_Box
            Envelope_Envelope_Intersect_Check
            Edge_Containment_Check
         Segment_Envelope_Violation_Check
            Get_Box
            Envelope_Regular_Segment_Intersect_Check
            Edge_Containment_Check
```

Figure 4-22 shows a PDL representation of the Horizontal Check algorithm.

```
ROUTINE Horizontal_Check;
PARAMETERS SUBJECT_SEGMENT IN, NOMINEE_SEGMENT IN, Subject_Fl_Id IN,
   Nominee_Fl_Id IN, Time_Overlap_Min IN, Time_Overlap_Max IN,
   Status OUT, Advisory_Time_Viol_Start OUT, Advisory_Time_Viol_End
   OUT, Priority_Time_Viol_Start OUT, Priority_Time_Viol_End OUT,
   Time_Msep OUT, Msep_Dist OUT;
DEFINED IN GLOSSARY
   Subject_Fl_Id
   Nominee_Fl_Id
   Time_Overlap_Min
   Time_Overlap_Max
   Advisory_Time_Viol_Start
   Advisory_Time_Viol_End
   Priority_Time_Viol_Start
   Priority_Time_Viol_End
   Time_Msep
   Msep_Dist
   SUBJECT_SEGMENT
   NOMINEE_SEGMENT;
DEFINE VARIABLES
   Status          Variable indicating whether or not the subject and
                     nominee aircraft violate the advisory horizontal
                     separation criterion;
IF SUBJECT_SEGMENT.first_cusp_type EQ 'regular' AND
   NOMINEE_SEGMENT.first_cusp_type EQ 'regular'
THEN
   CALL Regular_Segment_Horizontal_Check (SUBJECT_SEGMENT
      IN, NOMINEE_SEGMENT IN, Time_Overlap_Min IN, Time_Overlap_Max
      IN, Status OUT, Advisory_Time_Viol_Start OUT, Advisory_Time_
      Viol_End OUT, Priority_Time_Viol_Start OUT, Priority_Time_
      Viol_End OUT, Time_Msep OUT, Msep_Dist OUT);
ELSE
   CALL Maneuver_Envelope_Horizontal_Check (SUBJECT_SEGMENT IN,
      NOMINEE_SEGMENT IN, Subject_Fl_Id IN, Nominee_Fl_Id IN, Time_
      Overlap_Min IN, Time_Overlap_Max IN, Status OUT, Advisory_
      Time_Viol_Start OUT, Advisory_Time_Viol_End OUT, Priority_
      Time_Viol_Start OUT, Priority_Time_Viol_End OUT, Time_Msep
      OUT, Msep_Dist OUT);
END Horizontal_Check;
```

FIGURE 4-22
HORIZONTAL CHECK

Regular Segment Horizontal Check

The Regular Segment Horizontal Check algorithm uses the relative velocity and relative position vectors of the aircraft over their segments to identify violations of the horizontal separation criteria and to calculate the parameters which describe these violations. It invokes a routine called Relative Vectors which calculates the relative velocity and relative position vectors of the two aircraft. It then uses these relative vectors to compute the three coefficients which define the separation distance function (see Appendix B for the mathematical derivation of all of the parameters in the Regular Segment Horizontal Check). The algorithm calls the routine Violation Times, supplying it with the separation distance function coefficents and the advisory horizontal separation criterion Advisory_Seph. Violation Times determines if a violation of this criterion exists and, if so, calculates the start and end time of the violation. Under this circumstance, the routine is called again with the priority separation distance criterion, Priority_Seph, serving as an input.

The final portion of the algorithm calculates the time of minimum separation in the (x,y) plane, Time_Msep, and the minimum separation distance, Msep_Dist, using the formulas derived in Appendix B. Time_Msep is replaced by a bound of the common time interval if it falls outside of this interval. Specifically, if Time_Msep is less than Time_Overlap_Min or greater than Time_Overlap_Max, then its value is set equal to Time_Overlap_Min or Time_Overlap_Max, respectively, since, under such circumstances, the "true" time of minimum separation is one of the common time interval's endpoints (Figure 4-23 illustrates the need for this reassignment of values). With the time of minimum separation calculated, the algorithm finally computes Msep_Dist by simply substituting Time_Msep into the separation distance function.

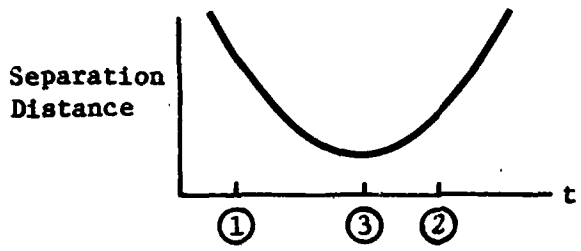
Figure 4-24 shows the PDL representation of the Regular Segment Horizontal Check Algorithm.

Relative Vectors

This element of the Regular Segment Horizontal Check calculates the horizontal relative velocity and the relative position vectors of the subject and nominee aircraft over their segments. It first computes the horizontal velocities of the subject aircraft and the nominee within their respective segments using the information provided in the cusps. The relative velocity vector, Rel_Vel, is then obtained by subtracting the nominee

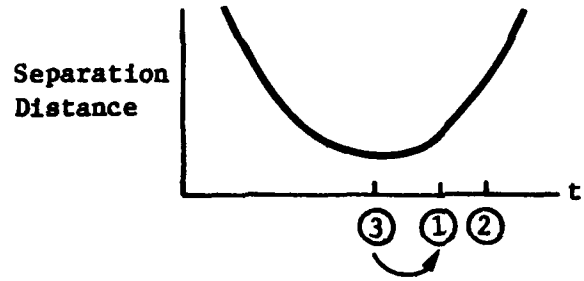
CASE 1: Time_Overlap_Min $\leq$ Time_Msep $\leq$ Time_Overlap_Max

Consequence: Time_Msep is left unchanged



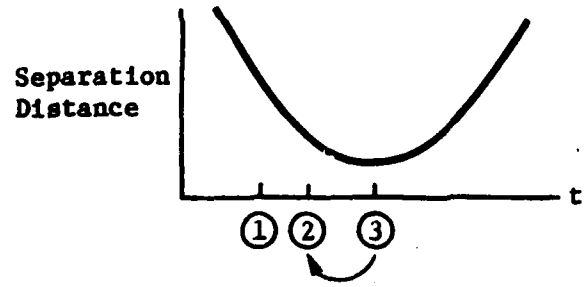
CASE 2: Time_Msep $<$ Time_Overlap_Min

Consequence: Time_Msep set equal to Time_Overlap_Min



CASE 3: Time_Msep $>$ Time_Overlap_Max

Consequence: Time_Msep set equal to Time_Overlap_Max



Legend:

1 = Time_Overlap_Min
2 = Time_Overlap_Max
3 = Time_Msep

FIGURE 4-23
DERIVATION OF TIME OF MINIMUM SEPARATION

```
ROUTINE Regular_Segment_Horizontal_Check;
PARAMETERS (SUBJECT_SEGMENT IN, NOMINEE_SEGMENT IN, Time_Overlap_Min
   IN, Time_Overlap_Max IN, Status OUT, Advisory_Time_Viol_Start OUT,
   Advisory_Time_Viol_End OUT, Priority_Time_Viol_Start OUT,
   Priority_Time_Viol_End OUT, Time_Msep OUT, Msep_Dist OUT);
REFER TO GLOBAL Advisory_Seph IN, Priority_Seph IN;
DEFINE IN GLOSSARY
   Time_Overlap_Min
   Time_Overlap_Max
   Advisory_Time_Viol_Start
   Advisory_Time_Viol_End
   Priority_Time_Viol_Start
   Priority_Time_Viol_End
   Time_Msep
   Msep_Dist
   SUBJECT_SEGMENT
   NOMINEE_SEGMENT;
DEFINE VARIABLES
   Status       Variable indicating whether or not the subject and
                   nominee aircraft violate the advisory horizontal
                   separation criterion
   A            Coefficient of the quadratic term in the separation
                   distance function (see Appendix B)
   B            Coefficient of the linear term in the separation
                   distance function (see Appendix B)
   C            Constant term in the separation distance function
                   (see Appendix B)
   Rel_Vel      Relative velocity of the subject and nominee aircraft
                   in the horizontal plane (i.e., subject aircraft
                   velocity minus nominee aircraft velocity)
      x            X component of the relative velocity
      y            Y component of the relative velocity
   Rel_Pos      Relative position of the subject and nominee aircraft
                   at Time_Overlap_Min in the horizontal plane (i.e.,
                   subject aircraft position minus nominee aircraft
                   position)
      x            X component of the relative position
      y            Y component of the relative position
   State        Variable indicating whether or not the subject and
                   nominee violate the specific horizontal separation
                   criterion under consideration
   Delta_T      Length of time between Time_Overlap_Min and the time
                   of minimum separation;
```

FIGURE 4-24
REGULAR SEGMENT HORIZONTAL CHECK

```
# calculate the relative vectors needed by this routine #
CALL Relative_Vectors (SUBJECT_SEGMENT IN, NOMINEE_SEGMENT IN,
   Time_Overlap_Min IN, Rel_Vel OUT, Rel_Pos OUT);
# calculate the coefficients of the separation distance function #
A = MAGNITUDE(Rel_Vel) ** 2;
B = 2 * DOT(Rel_Vel, Rel_Pos);
C = MAGNITUDE(Rel_Pos) ** 2;
# determine if the advisory horizontal separation criterion is  #
# violated and, if so, calculate the start and end times of the #
# violation                                                     #
CALL Violation_Times (A IN, B IN, C IN, Advisory_Seph IN, Time_
   Overlap_Min IN, Time_Overlap_Max IN, State OUT, Advisory_Time_
   Viol_Start OUT, Advisory_Time_Viol_End OUT);
IF State EQ 'violation'
THEN
   Status = 'Violation of advisory horizontal separation criterion';
   # given that a violation of the advisory horizontal separation #
   #  criterion has been detected, determine if the priority      #
   #  separation criterion is violated and, if so, calculate the  #
   #  start and end times of the violation                        #
   CALL Violation_Times (A IN, B IN, C IN, Priority_Seph IN,
      Time_Overlap_Min IN, Time_Overlap_Max IN, State OUT, Priority_
      Time_Viol_Start OUT, Priority_Time_Viol_End OUT);
   # calculate time of minimum separation and minimum separation #
   # distance of the two aircraft in the horizontal plane        #
   Time_Msep = (-B / (2 * A)) + Time_Overlap_Min;
   IF Time_Msep LT Time_Overlap_Min
   THEN
      Time_Msep = Time_Overlap_Min;
   ELSE
      IF Time_Msep GT Time_Overlap_Max
      THEN
         Time_Msep = Time_Overlap_Max;
   Delta_T = Time_Msep - Time_Overlap_Min;
   Msep_Dist = SQRT(A * Delta_T ** 2  +  B * Delta_T  +  C);
END Regular_Segment_Horizontal_Check;
```

FIGURE 4-24
REGULAR SEGMENT HORIZONTAL (Concluded)

velocity from the subject velocit Similarly, the relative position vector, Rel_Pos, is derived by first calculating the respective horizontal position vectors of the two aircraft at Time_Overlap_Min, the earliest time that the two segments overlap in time, and then finding the difference of the two vectors.

Figure 4-25 shows a PDL representation of the Relative Vectors Algorithm.

Violation Times

This element of the Regular Segment Horizontal Check determines whether or not the two aircraft violate the horizontal separation criterion, Seph, supplied as an argument to the routine and, if so, calculates the starting and ending times of the violation. The mathematical derivation of the relevant formulas is provided in Appendix B.

The algorithm computes the discriminant of an algebraic equation. This equation is obtained by subtracting the square of Seph from the square of the separation distance function and setting the result equal to zero. The nature of its roots indicate whether or not Seph is violated. If the discriminant is either a negative number or zero, the separation distance between the aircraft is equal to or exceeds Seph. Thus, no violation is expected to occur and the variable State is assigned a message indicating this. The parameters Time_Viol_Start and Time_Viol_End are set equal to a null value. If the discriminant is positive, then a violation is theoretically possible, given the magnitudes and directions of the relative vectors. Nevertheless, it is also possible that the violation of the separation criterion occurs outside of the time interval in common between the two aircraft. Whenever the derived earliest and latest time of violations indicate that the violation occurs outside the interval in common between the two aircraft, then State is assigned a message that there is no violation of the horizontal separation criterion. In this case, the parameters Time_Viol_Start and Time_Viol_End are set equal to a null value. Otherwise, State is made to indicate that there is a violation. If either of the two time bounds of the violation interval is located outside of the common time interval, say Time_Viol_Start is less than Time_Overlap_Min, then the corresponding bound of the common time interval becomes the new bound of the violation, that is (for the same example), Time_Overlap_Min becomes the new starting time of the violation period. The objective of this substitution is to

```
ROUTINE Relative_Vectors;
PARAMETERS SUBJECT_SEGMENT IN, NOMINEE_SEGMENT IN, Time_Overlap_Min
   IN, Rel_Vel OUT, Rel_Pos OUT;
DEFINED IN GLOSSARY
   Time_Overlap_Min
   SUBJECT_SEGMENT
   NOMINEE_SEGMENT;
DEFINE VARIABLES
   Rel_Vel              Relative velocity of the subject and nominee
                           aircraft in the horizontal plane (i.e.,
                           subject aircraft velocity minus nominee
                           aircraft velocity)
      x                    X component of the relative velocity
      y                    Y component of the relative velocity
   Rel_Pos              Relative position of the subject and
                           nominee aircraft at Time_Overlap_Min in
                           the horizontal plane (i.e., subject
                           aircraft position minus nominee
                           aircraft position)
      x                    X component of the relative position
      y                    Y component of the relative position
   Subject_Velocity     Horizontal velocity of the subject aircraft
                           within its segment
      x                    X component of the subject a/c velocity
      y                    Y component of the subject a/c velocity
   Nominee_Velocity     Horizontal velocity of the nominee aircraft
                           within its segment
      x                    X component of the nominee a/c velocity
      y                    Y component of the nominee a/c velocity
   Suoject_Position     Position of the subject aircraft in the
                           horizontal plane at time Time_Overlap_Min
      x                       X component of the subject a/c position
      y                       Y component of the subject a/c position
   Nominee_Position     Position of the nominee aircraft in the
                           horizontal plane at time Time_Overlap_Min
      x                       X component of the nominee a/c position
      y                       Y component of the nominee a/c
                                 position;
```

FIGURE 4-25
RELATIVE VECTORS

```
# calculate the subject aircraft velocity, the nominee aircraft #
# velocity, and the relative velocity of the two aircraft       #
Subject_Velocity = (SUBJECT_SEGMENT.second_xy_pair - SUBJECT_SEGMENT.
   first_xy_pair) / (SUBJECT_SEGMENT.second_t -
   SUBJECT_SEGMENT.first_t);
Nominee_Velocity = (NOMINEE_SEGMENT.second_xy_pair - NOMINEE_SEGMENT.
   first_xy_pair) / (NOMINEE_SEGMENT.second_t -
   NOMINEE_SEGMENT.first_t);
Rel_Vel = Subject_Velocity - Nominee_Velocity;
# calculate the subject aircraft position, the nominee aircraft #
# position, and the relative position of the two aircraft at    #
# Time_Overlap_Min                                              #
Subject_Position = SUBJECT_SEGMENT.first_xy_pair + (Time_Overlap_Min-
   SUBJECT_SEGMENT.first_t) * Subject_Velocity;
Nominee_Position = NOMINEE_SEGMENT.first_xy_pair + (Time_Overlap_Min-
   NOMINEE_SEGMENT.first_t) * Nominee_Velocity;
Rel_Pos = Subject_Position - Nominee_Position;
END Relative_Vectors;
```

FIGURE 4-25
RELATIVE VECTORS (Concluded)

define the true violation period and not the theoretical one obtained through straight calculations.

Figure 4-26 provides a PDL representation of the Violation Times Algorithm.

Maneuver Envelope Horizontal Check

The Maneuver Envelope Horizontal Check algorithm tests for violations of the FPCP horizontal separation criteria whenever the subject and/or the object aircraft are engaged in hold maneuvers or vertical maneuvers and previous tests for time overlap and vertical separation have not ruled out the possibility of an encounter. A test is first made to see if the advisory horizontal separation criterion, Advisory_Seph, is violated and if it is, a test for violation of the priority separation criterion is made. These tests are done by the Maneuver Envelope Test element. When either criterion is violated, the corresponding start and end times of violation (Advisory_Time_Viol_Start, Advisory_Time_Viol_End, and/or Priority_Time_Viol_Start, Priority_Time_Viol_End) are returned.

Figure 4-27 shows the PDL representation of the Maneuver Envelope Horizontal Check algorithm.

Maneuver Envelope Test

The Maneuver Envelope Test element performs the tests for violation of the horizontal separation criterion, Seph, when either the subject or the object aircraft is in a maneuver. Figure 4-28 illustrates the two holding pattern cases in which this algorithm is invoked and Figure 4-29 illustrates the vertical maneuver case. The horizontal criteria used are independent of the position of the aircraft within the maneuver. Violations depend only on the distance between the regions, in the horizontal plane, covered by the envelopes or segments involved. It is assumed that these regions are rectangles in the horizontal plane.

In the following discussion, the horizontal projection of a vertical maneuver will refer to the line segment in the horizontal plane corresponding to the two extreme time points of the maneuver (e.g., the right downstream and left upstream points).

When both the subject and object aircraft are involved in an airspace sweeping maneuver, a violation, in the horizontal plane, occurs in any of the following three cases:

ROUTINE Violation_Times;
PARAMETERS A IN, B IN, C IN, Seph IN, Time_Overlap_Min IN, Time_
Overlap_Max IN, State OUT, Time_Viol_Start OUT, Time_Viol_End OUT;
DEFINED IN GLOSSARY
Time_Overlap_Min
Time_Overlap_Max;
DEFINE VARIABLES

A	Coefficient of the quadratic term in the separation distance function (see Appendix B)
B	Coefficient of the linear term in the separation distance function (see Appendix B)
C	Constant term in the separation distance function (see Appendix B)
Seph	Horizontal separation criterion
State	Variable indicating whether or not the subject and nominee aircraft violate the specific horizontal separation criterion under consideration
Time_Viol_Start	Earliest time that the horizontal separation criterion is violated
Time_Viol_End	Latest time that the horizontal separation criterion is violated
Discriminant	Discriminant of the quadratic equation formed by setting the difference of the separation distance function squared and the horizontal separation criterion squared to zero (see Appendix B);

FIGURE 4-26
VIOLATION TIMES

```
Discriminant = B ** 2  -  4 * A * (C - Seph ** 2);
IF Discriminant LE 0
# equivalent to separation distance function  #
#  being greater than or equal to Seph        #
THEN
   State = 'no violation';
   Time_Viol_Start = NULL;
   Time_Viol_End = NULL;
ELSE
   # calculate the times of violation start and end #
   Time_Viol_Start = (-B - SQRT(Discriminant)) / (2 * A) +
      Time_Overlap_Min;
   Time_Viol_End = (-B + SQRT(Discriminant)) / (2 * A) +
      Time_Overlap_Min;
   # determine if the entire violation period occurs outside of the #
   # time interval common to both segments                          #
   IF Time_Viol_Start GT Time_Overlap_Max  OR  Time_Viol_End LT
      Time_Overlap_Min
   THEN
      State = 'no violation';
      Time_Viol_Start = NULL;
      Time_Viol_End = NULL;
   ELSE
      State = 'violation';
      # calculate the "true" times of violation start and end #
      Time_Viol_Start = MAX(Time_Viol_Start, Time_Overlap_Min);
      Time_Viol_End = MIN(Time_Viol_End, Time_Overlap_Max);
END Violation_Times;
```

FIGURE 4-26
VIOLATION TIMES (Concluded)

```
ROUTINE Maneuver_Envelope_Horizontal_Check;
#checks for violation of horizontal separation criterion when the#
   #segments are maneuver envelopes#
#tests for violation of both advisory and priority horizontal#
   #separation criterion are made; if a violation is detected#
   #returns the start and end times of the violation#
PARAMETERS SUBJECT_SEGMENT IN, NOMINEE_SEGMENT IN, Subject_Fl_Id IN,
   Nominee_Fl_Id IN, Time_Overlap_Min IN, Time_Overlap_Max IN,
   Status OUT, Advisory_Time_Viol_Start OUT, Advisory_Time_Viol_End
   OUT, Priority_Time_Viol_Start OUT, Priority_Time_Viol_End OUT,
   Time_Msep OUT, Msep_Dist OUT;
REFER TO GLOBAL Advisory_Seph, IN, Priority_Seph IN;
DEFINED IN GLOSSARY
   Subject_Fl_Id
   Nominee_Fl_Id
   Time_Overlap_Min
   Time_Overlap_Max
   Advisory_Time_Viol_Start
   Advisory_Time_Viol_End
   Priority_Time_Viol_Start
   Priority_Time_Viol_End
   Time_Msep
   Msep_Dist
   SUBJECT_SEGMENT
   NOMINEE_SEGMENT;
DEFINE VARIABLES
   Status      Variable indicating the outcome of a particular Fine
                  Filter test;
```

FIGURE 4-27
MANEUVER_ENVELOPE_HORIZONTAL_CHECK

```
Time_Msep = NULL;
Msep_Dist = NULL;
#first test for violation of advisory separation criterion#
CALL Maneuver_Envelope_Test (SUBJECT_SEGMENT IN, NOMINEE_SEGMENT IN,
   Subject_Fl_Id IN, Nominee_Fl_Id IN, Time_Overlap_Min IN, Time_
   Overlap_Max IN, Advisory_Seph IN, Status OUT, Advisory_Time_Viol_
   Start OUT, Advisory_Time_Viol_End OUT);
#test for priority violation only if an advisory violation has been#
   #detected#
IF Status EQ 'violation'
THEN #test for violation of priority separation criterion#
   Status = 'violation of advisory horizontal separation criterion';
   CALL Maneuver_Envelope_Test (SUBJECT_SEGMENT IN, NOMINEE_SEGMENT
      IN, Subject_Fl_Id IN, Nominee_Fl_Id IN, Time_Overlap_Min IN,
      Time_Overlap_Max IN, Priority_Seph IN, Status OUT, Priority_
      Time_Viol_Start OUT, Priority_Time_Viol_End OUT);
END Maneuver_Envelope_Horizontal_Check;
```

FIGURE 4-27
MANEUVER_ENVELOPE_HORIZONTAL_CHECK (Concluded)

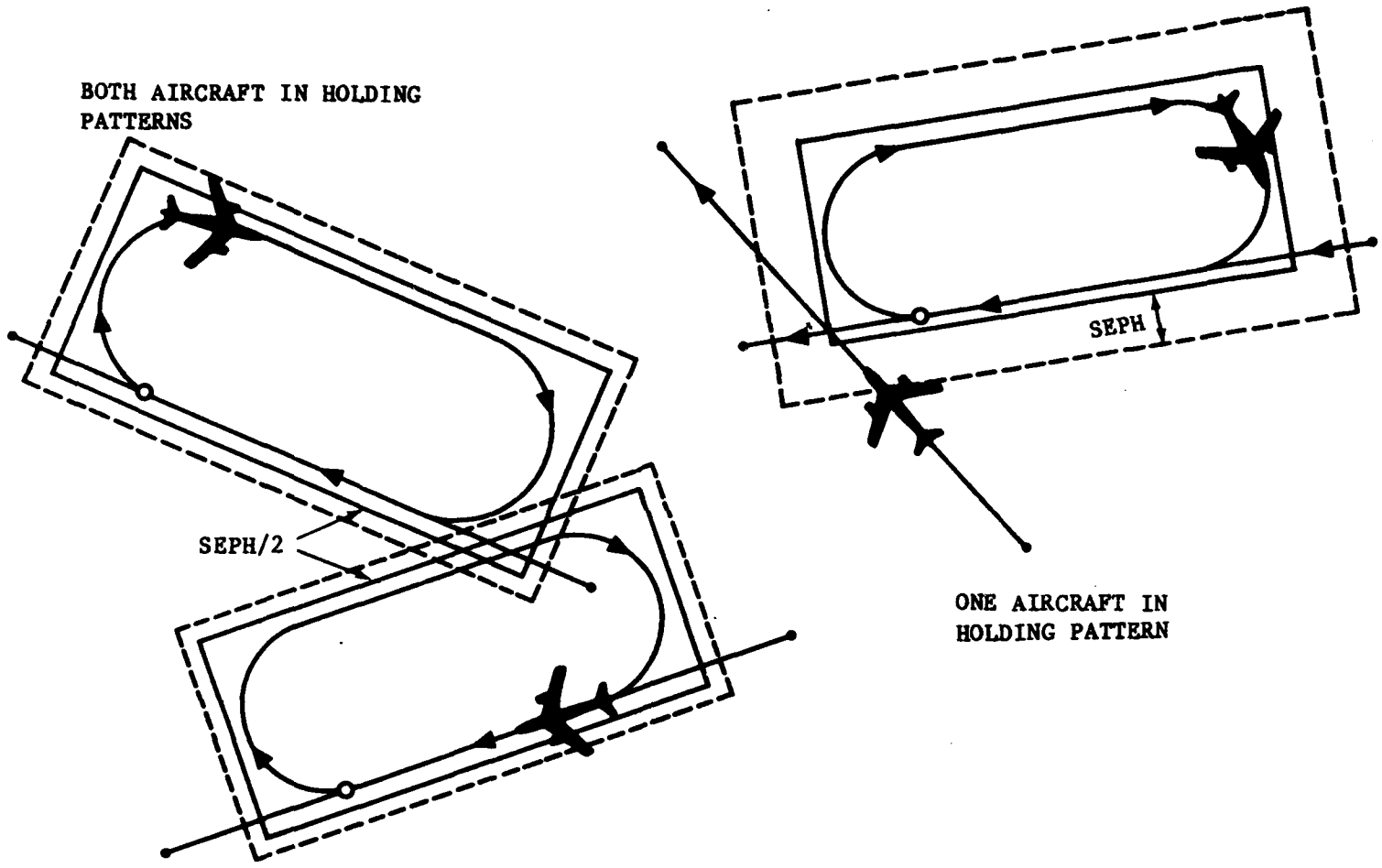
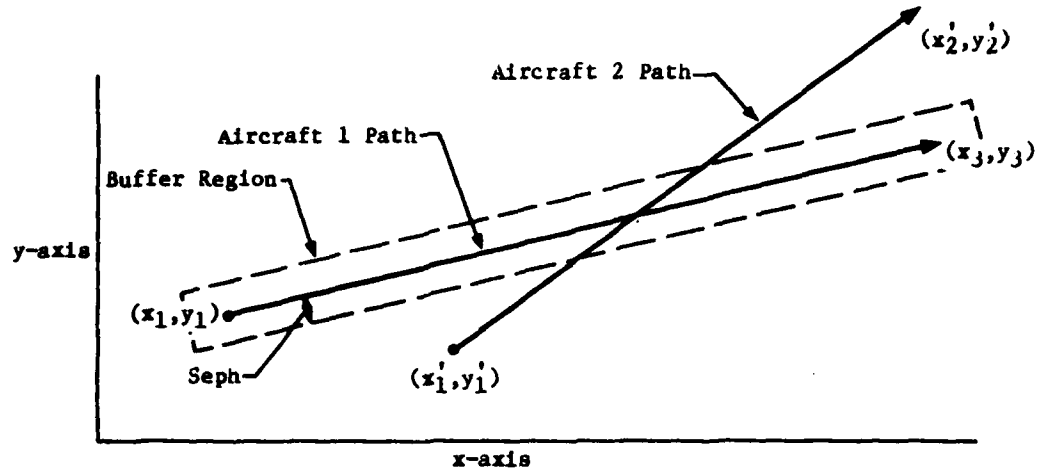
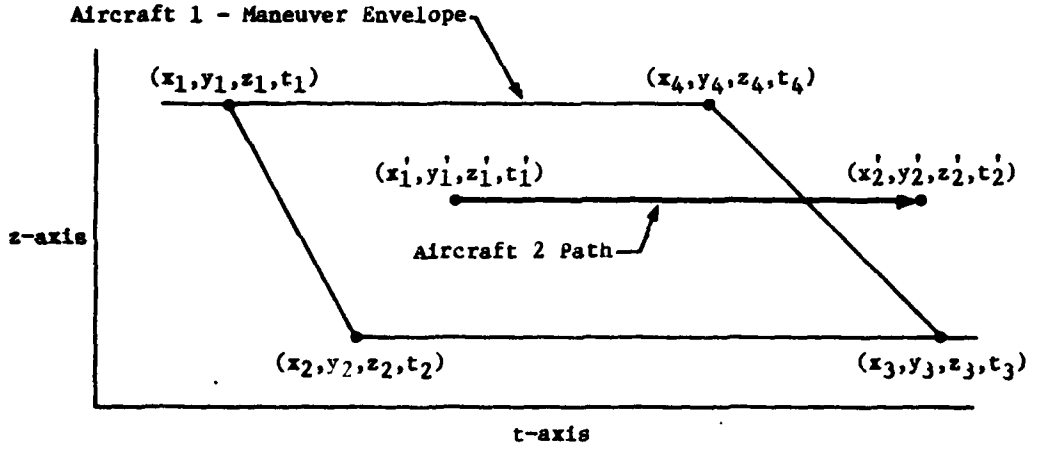


FIGURE 4-28
CASES WHERE THE HOLDING PATTERN HORIZONTAL CHECK IS INVOKED



(a) Aircraft Paths and Buffer Region in (x,y) Plane



(b) Vertical Maneuver Envelope of a First Aircraft and Path of Second Aircraft in (z,t) Plane

FIGURE 4-29
EXAMPLE OF HORIZONTAL CHECK FOR A VERTICAL MANEUVER

- the horizontal maneuver envelope of one of the aircraft is completely contained in the other
- the horizontal maneuver envelopes intersect
- the horizontal maneuver envelopes (closest points on each) are within a distance of (advisory or priority) Seph of each other

These cases are checked in the element Envelope Envelope Violation Check.

When one of the subject or object segments is regular, a violation occurs in any of the following three similar cases:

- the regular segment is contained within the horizontal maneuver envelope in the horizontal plane
- the regular segment intersects the horizontal maneuver envelope
- the distance (closest points on the segment and envelope) between the regular segment and the horizontal envelope are within a distance of (advisory or priority) Seph of each other in the horizontal plane

These cases are checked in the element Segment Envelope Violation Check.

The variable Seph is an input parameter, thus permitting this algorithm to be used for the testing of the violation of both the advisory and the priority horizontal separation criteria (that is, Advisory_Seph and Priority_Seph) by its calling routine, Maneuver Envelope Horizontal Check. Output from this routine are the variables Time_Viol_Start and Time_Viol_End, and a status variable indicating whether or not a violation has occurred. Whether the times of violation indicate advisory or priority times depends on which input value for Seph was used.

Figure 4-30 shows the PDL representation of the Maneuver Envelope Test element.

Envelope Envelope Violation Check

The Envelope Envelope Violation Check element tests for violation of the horizontal separation criterion whenever both aircraft are involved in maneuvers. The envelope of each aircraft is extended around its perimeter by one-half Seph

```
ROUTINE Maneuver_Envelope_Test;
#tests for violation of a horizontal separation criterion Seph#
   #whenever either of the two aircraft is in a maneuver envelope#
PARAMETERS SUBJECT_SEGMENT IN, NOMINEE_SEGMENT IN, Subject_Fl_Id IN,
   Nominee_Fl_Id IN, Time_Overlap_Min IN, Time_Overlap_Max IN, Seph
   IN, Status OUT, Time_Viol_Start OUT, Time_Viol_End OUT;
DEFINED IN GLOSSARY
   Subject_Fl_Id
   Nominee_Fl_Id
   Time_Overlap_Min
   Time_Overlap_Max
   Time_Viol_Start
   Time_Viol_End
   SUBJECT_SEGMENT
   NOMINEE_SEGMENT
DEFINE VARIABLES
   Seph     Horizontal separation criterion
   Status   Variable indicating the outcome of a particular Fine
               Filter test;
CHOOSE CASE
   WHEN SUBJECT_SEGMENT.first_cusp_type IS IN ('hold', 'vertical
      hold', 'vertical maneuver') AND NOMINEE_SEGMENT.first_cusp_
      type IS IN ('hold', 'vertical hold', 'vertical maneuver') THEN
      #check the two maneuver envelope case#
      CALL Envelope_Envelope_Violation_Check(SUBJECT_SEGMENT IN,
         NOMINEE_SEGMENT IN, Subject_Fl_Id IN, Nominee_Fl_Id IN,
         Time_Overlap_Min IN, Time_Overlap_Max In, Seph IN,
         Status OUT, Time_Viol_Start OUT, Time_Viol_End OUT);
   WHEN SUBJECT_SEGMENT.first_cusp_type IS IN ('hold', 'vertical
      maneuver') THEN #object in hold, nominee regular#
      CALL Segment_Envelope_Violation_Check(Subject_Fl_Id IN,
         SUBJECT_SEGMENT IN, NOMINEE_SEGMENT IN, Seph IN, Time_
         Overlap_Min IN, Time_Overlap_Max IN, Time_Viol_Start OUT,
         Time_Viol_End OUT, Status OUT);
   OTHERWISE #nominee in a maneuver envelope, subject is regular#
      CALL Segment_Envelope_Violation_Check(Nominee_Fl_Id IN,
         NOMINEE_SEGMENT IN, SUBJECT_SEGMENT IN, Seph IN, Time_
         Overlap_Min IN, Time_Overlap_Max IN, Time_Viol_Start OUT,
         Time_Viol_End OUT, Status OUT);
END Maneuver_Envelope_Test;
```

FIGURE 4-30
MANEUVER_ENVELOPE_TEST

(advisory and, in a subsequent call, priority), thus forming two temporary two-dimensional envelopes in the horizontal plane which are then compared to determine if a violation has occurred. This is done by the Get Box element. Once this extension is made, it is sufficient to test the two new extended envelopes for the following cases. Let A be the subject's extended envelope and let B be the object's extended envelope. Then the three cases to be tested are the following:

- Do the boundaries of A and B intersect?
- Is region A contained within the boundary of region B?
- Is region B contained within the boundary of region A?

Note that it is not necessary to test for closeness of regions A and B since this is already accounted for in the extensions made to the original envelopes.

The first case is accomplished by testing each edge of one envelope against each edge of the other. If any two edges intersect, the test ends and a violation message is set. If no intersection is determined, then the envelopes are tested to see if either is contained within the other. These tests are done in the Envelope Envelope Intersect Check element.

This case may be implemented by testing any edge (say the edge connecting the right downstream and right upstream vertices) of the boundary of region A against all edges of the boundary of region B. The element Edge Containment Check is called to perform the test for the selected edge being contained in the boundary of region B.

If this test does not produce a violation, then the next case is tested. The test for region B being contained within the boundary of region A is accomplished by reversing the roles of A and B in the discussion above. If a violation is determined, then the times of violation start and end are set equal to the minimum and maximum times of the overlap period, respectively.

Figure 4-31 illustrates a PDL representation of the Envelope Envelope Violation Check element.

Get Box

The element Get Box builds a rectangular box around the horizontal projection of the maneuver. Using the input parameter S as the separation distance, the algorithm extends the perimeter of a holding pattern by this distance. In the case of a vertical maneuver the extension is made by forming a

```
ROUTINE Envelope_Envelope_Violation_Check;
#tests for violation of a horizontal separation criterion Seph#
   #whenever both aircraft are in maneuvers, (either holds or#
   #vertical maneuvers)#
PARAMETERS SUBJECT_SEGMENT IN, NOMINEE_SEGMENT IN, Subject_Fl_Id IN,
   Nominee_Fl_Id IN, Time_Overlap_Min IN, Time_Overlap_Max IN, Seph
   IN, Status OUT, Time_Viol_Start OUT, Time_Viol_End OUT;
DEFINED IN GLOSSARY
   Subject_Fl_Id
   Nominee_Fl_Id
   Time_Overlap_Min
   Time_Overlap_Max
   Time_Viol_Start
   Time_Viol_End
   SUBJECT_SEGMENT
   NOMINEE_SEGMENT;
DEFINE VARIABLES
   Seph     Horizontal separation criterion
   Status   Variable indicating the outcome of a particular Fine
               Filter test;
DEFINE TABLES
   SUBJECT_BOX_VERTICES     Vertices in horizontal plane of the
                               subject box surrounding the maneuver
                               envelope
      right_downstream_vertex
         x                        x coordinate
         y                        y coordinate
      right_upstream_vertex
         x                        x coordinate
         y                        y coordinate
      left_downstream_vertex
         x                        x coordinate
         y                        y coordinate
      left_upstream_vertex
         x                        x coordinate
         y                        y coordinate
      edge1 AGGREGATE(right_downstream_vertex, right_upstream_vertex)
      edge2 AGGREGATE(right_upstream_vertex, left_upstream_vertex)
      edge3 AGGREGATE(left_upstream_vertex, left_downstream_vertex)
      edge4 AGGREGATE(left_downstream_vertex, right_downstream_
         vertex)
   NOMINEE_BOX_VERTICES     Vertices in the horizontal plane
                               of the box surrounding the nominee
                               maneuver envelope; fields defined like
                               SUBJECT_BOX_VERTICES;
```

FIGURE 4-31
ENVELOPE_ENVELOPE_VIOLATION_CHECK

```
#check the two maneuver envelope case#
#get the coordinates for the subject, extended by a distance of#
   #Seph/2 surrounding the horizontal hold or the horizontal segment#
   #of the vertical maneuver#
CALL Get_Box(Subject_Fl_Id IN, SUBJECT_SEGMENT.first_t IN, Seph/2
   IN, SUBJECT_SEGMENT.first_cusp_type IN, SUBJECT_BOX_VERTICES OUT,
   Time_Overlap_Min INOUT, Time_Overlap_Max INOUT);
#get the coordinates for the nominee, extended by a distance of#
   #Seph/2 surrounding the horizontal hold or the horizontal segment#
   #of the vertical maneuver#
CALL Get_Box(Nominee_Fl_Id IN, NOMINEE_SEGMENT.first_t IN, Seph/2
   IN, NOMINEE_SEGMENT.first_cusp_type IN, NOMINEE_BOX_VERTICES OUT,
   Time_Overlap_Min INOUT, Time_Overlap_Max INOUT);
#first test if the segments intersect#
CALL Envelope_Envelope_Intersect_Check(SUBJECT_BOX_VERTICES IN,
   NOMINEE_BOX_VERTICES IN, Status OUT);
IF Status EQ 'no violation'
THEN #they do not intersect#
   #test if subject envelope is contained within nominee envelope#
      #need only check one edge of the hold since they don't#
      #intersect take the edge connected by the vertices right_#
      #downstream_vertex and right_upstream_vertex#
   CALL Edge_Containment_Check(SUBJECT_BOX_VERTICES.Edge1 IN,
      NOMINEE_BOX_VERTICES IN, Status OUT);
   IF Status EQ 'no violation'
   THEN #subject box is not contained within the nominee's#
      #test if the nominee envelope is contained within the#
         #subject's#
      CALL Edge_Containment_Check(NOMINEE_BOX_VERTICES.Edge1 IN,
         SUBJECT_BOX_VERTICES IN, Status OUT);
IF Status EQ 'violation'
THEN
   #set the time of violation to the overlap times#
   Time_Viol_Start = Time_Overlap_Min;
   Time_Viol_End = Time_Overlap_Max;
END Envelope_Envelope_Violation_Check;
```

FIGURE 4-31
ENVELOPE_ENVELOPE_VIOLATION_CHECK (Concluded)

box a distance of S around the line segment connecting the right downstream and left upstream points of the envelope in the (x,y) plane. Also in this case the Time_Overlap_Min and Time_Overlap_Max are reset to coincide with the time coordinates of these points.

Figure 4-32 illustrates a PDL version of the Get Box algorithm.

Envelope Envelope Intersect Check

The Envelope Envelope Intersect Check element tests the extended box around the subject's envelope to see if it intersects the extended box around the object's envelope. This is accomplished by testing each edge of the subject against each edge of the object iteratively until a violation is detected. If a violation is detected, the flag, Status, is set to a violation status.

Figure 4-33 illustrates a PDL version of the Envelope Envelope Intersect Check element.

Edge Containment Check

The Edge Containment Check element takes as input the vertices of a box in the horizontal plane (BOX_VERTICES) and the vertices of an edge (Edge) and tests to see if Edge is completely contained within the rectangle defined by BOX_VERTICES. This is accomplished by first determining the points of intersection of the line passing through the vertices of Edge and the rectangle defined by BOX_VERTICES. Once this is done, if there is an intersection, then the line segment defined by Edge is tested to see if the x coordinates and y coordinates of Edge are within the intervals defined by the x and y coordinates of the intersection points. If they are, then Edge is completely contained in the rectangular region defined by BOX_VERTICES and the flag, Status, is set to a violation.

Figure 4-34 illustrates a PDL version of the Edge Containment Check algorithm.

Segment Envelope Violation Check

If either of the segments to be tested is a regular segment, then the horizontal envelope of the other is extended around its perimeter by Seph miles to provide a buffer to guarantee sufficient separation. This is done by the element Get Box, described above. A similar test to the ones above is then

```
ROUTINE Get_Box;
PARAMETERS Fl_Id IN, T IN, S IN, Cusp_Type IN, BOX_VERTICES OUT
   Time_Overlap_Min INOUT, Time_Overlap_Max INOUT;
REFER TO GLOBAL MANEUVER_ENVELOPES IN;
DEFINED IN GLOSSARY
   Time_Overlap_Min
   Time_Overlap_Max;
DEFINE VARIABLES
   Fl_Id        Flight plan identifier of aircraft in hold
   T            Time of entry into hold
   S            Separation criterion
   Cusp_Type    Type of cusp associated with this segment;
DEFINE TABLES
   BOX_VERTICES              Vertices of the extended box; fields
                                defined like SUBJECT_BOX_VERTICES in
                                ROUTINE Envelope_Envelope_Violation
                                Check
   MANEUVER_ENVELOPE_TEMP    Temporary copy of one record of MANEUVER_
                                ENVELOPES, stored as a table; fields
                                defined like MANEUVER_ENVELOPES
      vert_edge AGGREGATE(lu_x,lu_y,rd_x,rd_y) #vertices of edge#
         #defined by the start and end of the envelope in the#
            #horizontal plane associated with the minimum and#
            #maximum time in the envelope#;
MANEUVER_ENVELOPE_TEMP = SELECT FIELDS ALL
   FROM MANEUVER_ENVELOPES
   WHERE MANEUVER_ENVELOPES.Fl_id EQ Fl_Id AND MANEUVER_ENVELOPES.
      time EQ T;
IF Cusp_Type EQ 'hold' OR Cusp_Type EQ 'vertical hold'
THEN #hold case#
   BOX_VERTICES = MANEUVER_ENVELOPE_TEMP;
   Calculate the coordinates of the extended holding pattern for the
      holding pattern with vertices BOX_VERTICES extending it by a
      distance S around its perimeter;
ELSE #vertical maneuver envelope#
   Calculate the coordinates of the box around the segment
      MANEUVER_ENVELOPE_TEMP.vert_edge a distance of S surrounding
      the segment;
   #override the values of the overlap times by the endpoint times#
      #of the vertical maneuver, given by the right#
      #downstream and left upstream vertices#
   Time_Overlap_Min = MANEUVER_ENVELOPE_TEMP.lu_t;
   Time_Overlap_Max = MANEUVER_ENVELOPE_TEMP.rd_t;
Store the resulting extended box vertices back in BOX_VERTICES;
END Get_Box;
```

FIGURE 4-32
GET_BOX

```
ROUTINE Envelope_Envelope_Intersect_Check;
PARAMETERS SUBJECT_BOX_VERTICES IN, NOMINEE_BOX_VERTICES IN, Status
   OUT;
DEFINE VARIABLES
   Status   Variable indicating the outcome of a particular Fine
            Filter test;
DEFINE TABLES
   SUBJECT_BOX_VERTICES     Vertices of the box surrounding the
                               maneuver envelope of the subject in
                               the horizontal plane; fields defined
                               like SUBJECT_BOX_VERTICES in ROUTINE
                               Envelope_Envelope_Violation_Check
   NOMINEE_BOX_VERTICES     Vertices of the box surrounding the
                               maneuver envelope of the nominee in
                               the horizontal plane; fields defined
                               like SUBJECT_BOX_VERTICES;
Status = 'no violation';
REPEAT UNTIL Status EQ 'violation' OR all edges of SUBJECT_BOX_
   VERTICES have been tested;
   Select the next edge from SUBJECT_BOX_VERTICES;
   REPEAT UNTIL Status EQ 'violation' OR all edges of NOMINEE_BOX_
      VERTICES have been tested;
      Select next edge from NOMINEE_BOX_VERTICES;
      IF the edges selected intersect in the horizontal plane
      THEN
         Status = 'violation';
   END Envelope_Envelope_Intersect_Check;
```

FIGURE 4-33
ENVELOPE_ENVELOPE_INTERSECT_CHECK

ROUTINE Edge_Containment_Check;
PARAMETERS REGULAR_SEGMENT IN, BOX_VERTICES IN, Status OUT;
DEFINE VARIABLES

Status	Variable indicating whether or not the subject and nominee segments violate the horizontal separation criterion
Box_Test_Vertex(4)	Array containing vertices of the box
Regular_Line_Coeff(3)	Coefficients of the line through the regular segment
Box_Line_Coeff(3)	Coefficients of the line through an edge of the box
I	Index tracking sides of the box
J	Number of intersecting points between line and lines of the box
X1	Minimum intersection x coordinate
X2	Maximum intersection x coordinate
Y1	Minimum intersection y coordinate
Y2	Maximum intersection y coordinate
Test_Point	Variable used to test point of intersection
X	x coordinate
Y	y coordinate
Int_Pt(2)	Points of intersection, if they intersect end points of overlap of edges if edges coincident with an edge of the box
X	x coordinate
Y	y coordinate;

DEFINE TABLES

REGULAR_SEGMENT	Pair of cusps representing the regular segment being processed; fields defined like SUBJECT SEGMENT in Glossary
BOX_VERTICES	Vertices of the box surrounding the manuever envelope in the horizontal plane; fields defined like SUBJECT_BOX_VERTICES in ROUTINE Envelope_Envelope_Horizontal_Check;

FIGURE 4-34
EDGE_CONTAINMENT_CHECK

```
Status = 'no violation';
Box_Test_Vertex(1) = BOX_VERTICES.right_downstream_vertex;
Box_Test_Vertex(2) = BOX_VERTICES.right_upstream_vertex;
Box_Test_Vertex(3) = BOX_VERTICES.left_downstream_vertex;
Box_Test_Vertex(4) = BOX_VERTICES.left_downstream_vertex;
#determine equation of the line connecting vertices of#
   #REGULAR_SEGMENT in the xy plane#
Regular_Line_Coeff = LINE(REGULAR_SEGMENT.hz_first_vtx,
   REGULAR_SEGMENT.hz_sec_vtx);
I = 1; J = 0;
#determine points of intersection of line coincident to Edge and the#
   #edge of BOX_VERTICES#
REPEAT UNTIL I GT 4 OR J GE 2
   #test Ith edge#
   Box_Line_Coeff = LINE(Box_Test_Vertex(I), Box_Test_Vertex(I+1));
   IF the lines Box_Line_Coeff and Regular_Line_Coeff intersect
   THEN
      #get intersection point#
      Test_Point = Box_Line_Coeff INTERSECTION Regular_Line_Coeff;
      #test if the point lies on the edge of the box#
      X1 = MIN(Box_Test_Vertex(I).X, Box_Test_Vertex(I+1).X);
      X2 = MAX(Box_Test_Vertex(I).X, Box_Test_Vertex(I+1).X);
      Y1 = MIN(Box_Test_Vertex(I).Y, Box_Test_Vertex(I+1).Y);
      Y2 = MAX(Box_Test_Vertex(I).Y, Box_Test_Vertex(I+1).Y);
      IF Test_Point.X is in the open interval (X1,X2) AND
         Test_Point.Y is in the open interval (Y1,Y2)
      THEN #lies on the edge#
         J = J + 1;
         Int_Pt(J) = Test_Point;
   #check for containment within the box#
   IF J NE 0 the lines intersect#
   THEN #test if Edge is contained in the box by checking if the#
      #line segment#
      X1 = MIN(Int_Pt(1).X, Int_Pt(2).X);
      X2 = MAX(Int_Pt(1).X, Int_Pt(2).X);
      Y1 = MIN(Int_Pt(1).Y, Int_Pt(2).Y);
      Y2 = MAX(Int_Pt(1).Y, Int_Pt(2).Y);
      IF the projection of REGULAR_SEGMENT on the x-axis is in the
         interval [X1, X2] AND the projection of REGULAR_SEGMENT on
         the y-axis is in the closed interval [Y1, Y2]
      THEN
         Status = 'violation';
      I = I + 1; #next edge#
END Edge_Containment_Check;
```

FIGURE 4-34
EDGE_CONTAINMENT_CHECK (Concluded)

performed. First, the regular segment is tested to see if it intersects the extended envelope in the horizontal plane. This test is performed by the element Segment Envelope Intersect Check. If they intersect, then a violation exists and the appropriate status message is set. In this case, the time of violation start is set equal to the maximum of the time of overlap start and the minimum of the intersection times. The time of violation end is set equal to the minimum of the time of overlap end and the maximum of the intersection times. If the intersection occurs at a single point, then the end and start time of violation are equal.

If no violation is detected, then the segment is tested to check whether it is completely contained within the region of the extended horizontal envelope. This is accomplished by calling the element Edge Containment Check, described above. If a violation is detected, then an appropriate message is set and the time of violation start and end are set equal to the time of overlap minimum and maximum, respectively.

Figure 4-35 illustrates a PDL version of the Segment Envelope Violation Check element.

Segment Envelope Intersect Check

The Segment Envelope Intersect Check element tests a regular segment of an aircraft and the extended rectangular region about the horizontal projection of the envelope for intersection. Each edge of the rectangular region is tested against the regular segment to see if they are coincident and overlap and if they intersect. If this is the case, testing ends with a violation status and the points of overlap are determined. If not, the edge of the rectangular region is tested for intersection with the regular segment and the points of intersection are determined.

In the case of a violation, the times of violation are computed by first computing the times associated with the points of intersection and then determining the maximum between the Time_Overlap_Min and the first time intersect in the case of the Time_Viol_Start and determining the minimum between the Time_Overlap_Max and the second time intersect in the case of Time_Viol_End. The times associated with the intersection points are determined by interpolating using the fact that the velocity is assumed constant for a given aircraft.

Figure 4-36 illustrates a PDL version of the Segment Envelope Intersect Check algorithm.

```
ROUTINE Segment_Envelope_Violation_Check;
   PARAMETERS Env_Fl_Id IN, ENVELOPE_SEGMENT IN, REGULAR_SEGMENT IN,
      Seph IN, Time_Overlap_Min IN,Time_Overlap_Max IN, Time_Viol_
      Start OUT, Time_Viol_End OUT, Status OUT;
DEFINED IN GLOSSARY
   Nominee_Fl_Id
   Time_Overlap_Min
   Time_Overlap_Max
   Time_Viol_Start
   Time_Viol_End;
DEFINE VARIABLES
   Env_Fl_Id   Flight identifier of maneuver envelope segment
   Status      Variable indicating the outcome of a paricular Fine
                  Filter test
   Seph        Horizontal separation criterion;
DEFINE TABLES
   ENVELOPE_SEGMENT  Pair of cusps representing the segment being
                        processed for the segment that is in a
                        maneuver envelope; fields defined like
                        SUBJECT_SEGMENT in Glossary
   REGULAR_SEGMENT   Pair of cusps representing the segment
                        being processed for the regular segment;
                        fields defined like ENVELOPE_SEGMENT
   BOX_VERTICES      Vertices of the box surrounding the maneuver
                        envelope in the horizontal plane; fields
                        defined like SUBJECT_BOX_VERTICES in ROUTINE
                        Envelope_Envelope_Horizontal_Check;
```

FIGURE 4-35
SEGMENT_ENVELOPE_VIOLATION_CHECK

```
#get the coordinates for the nominee hold extended a distance#
   #of Seph about its perimeter#
CALL Get_Box(Env_Fl_Id IN, ENVELOPE_SEGMENT.first_cusp_t IN,
   Seph IN, ENVELOPE_SEGMENT.first_cusp_type IN, BOX_VERTICES OUT,
   Time_Overlap_Min INOUT, Time_Overlap_Max INOUT);
#test if the box around the envelope and the requlars segment#
   #intersect#
CALL Segment_Envelope_Intersect_Check(BOX_VERTICES IN, REGULAR_
   SEGMENT IN, Time_Overlap_Min IN, Time_Overlap_Max IN, Time_Viol_
   Start OUT, Time_Viol_End OUT, Status OUT);
IF Status EQ 'no violation'
THEN
   #check if regular segment is contained in box#
   CALL Edge_Containment_Check(REGULAR_SEGMENT IN, BOX_VERTICES IN,
      Status OUT);
   IF Status EQ 'violation'
   THEN
      Time_Viol_Start = Time_Overlap_Min;
      Time_Viol_End = Time_Overlap_Max;
END  Segment_Envelope_Violation_Check;
```

FIGURE 4-35
SEGMENT_ENVELOPE_VIOLATION_CHECK (Concluded)

```
ROUTINE Segment_Envelope_Intersect_Check;
PARAMETERS BOX_VERTICES IN, SEGMENT IN, Time_Overlap_Min IN, Time_
   Overlap_Max IN, Time_Viol_Start OUT, Time_Viol_End OUT, Status
   OUT;
DEFINED IN GLOSSARY
   Time_Overlap_Min
   Time_Overlap_Max
   Time_Viol_Start
   Time_Viol_End;
DEFINE VARIABLES
   Status                 Variable indicating the outcome of a
                             particular Fine Filter test
   Int_Pt(2)              Points in x,y,t of intersection between the
                             box and segment
      T                      t coordinate
      Hz_comp                Horizontal coordinates
         X                      x coordinate
         Y                      y coordinate
   Test_Box_Edges(4)      Array of edges used in testing for
                             intersection
      first_x                x coordinate of first vertex
      first_y                y coordinate of first vertex
      second_x               x coordinate of second vertex
      second_y               y coordinate of second vertex
   Line_Segment_Coeff     Coefficients of the equation of the line
                             passing through all points of the segment
                             named SEGMENT
   Line_Box_Coeff         Coefficients of the equation of the line
                             passing through all points of an edge of
                             the box defined by the vertices BOX
                             VERTICES
   I, J                   Indices for looping;
DEFINE TABLES
   BOX_VERTICES   Vertices of the box surrounding the manuever
                     envelope of the segemnt being considered;
                     fields defined like SUBJECT_BOX_VERTICES in
                     ROUTINE Envelope_Envelope_Horizontal_Check
   SEGMENT        Pair of cusps representing the regular segment
                     being considered; fields defined like SUBJECT
                     SEGMENT in the Glossary;
```

FIGURE 4-36
SEGMENT_ENVELOPE_INTERSECT_CHECK

```
Test_Box_Edge(1) = BOX_VERTICES.edge1; #temporary array of box edges#
Test_Box_Edge(2) = BOX_VERTICES.edge2;
Test_Box_Edge(3) = BOX_VERTICES.edge3;
Test_Box_Edge(4) = BOX_VERTICES.edge4;
Status = 'no violation';
I = 1; J = 0;
#determine the coefficients of the line passing through SEGMENT#
Line_Segment_Coeff = LINE(SEGMENT.hz_first_vtx, SEGMENT.hz_sec_vtx);
#determine the points of intersection#
REPEAT UNTIL J EQ 2 OR all edges of BOX_VERTICES have been tested;
   Line_Box_Coeff = LINE(Test_Box_Edge(I)); #line through Ith edge#
   IF Line_Box_Coeff EQ Line_Segment_Coeff AND the edges (SEGMENT.hz_
      first_vtx, SEGMENT.hz_sec_vtx) and Test_Box_Edges(I) overlap
   THEN #lines coincide, get intersect end pts#
      Status = 'violation';
      Int_Pt(1).X = MAX(MIN(SEGMENT.first_x, SEGMENT.second_x),
         MIN(Test_Box_Edges(I).first_x, Test_Box_Edges(I).second_x));
      Int_Pt(2).X = MIN(MAX(SEGMENT.first_x, SEGMENT.second_x),
         MAX(Test_Box_Edges(I).first_x, Test_Box_Edges(I).second_x));
      Int_Pt(1).Y = MAX(MIN(SEGMENT.first_y, SEGMENT.second_y),
         MIN(Test_Box_Edges(I).first_y, Test_Box_Edges(I).second_y));
      Int_Pt(2).Y = MIN(MAX(SEGMENT.first_y, SEGMENT.second_y),
         MAX(Test_Box_Edges(I).first_y, Test_Box_Edges(I).second_y));
   ELSE #test for unique intersect point#
      IF SEGMENT and Test_Box_Edge(I) intersect in the (x,y) plane
      THEN
         J = J + 1;
         Status = 'violation';
         Int_Pt(J) = Line_Box_Coeff INTERSECTION Line_Segment_Coeff;
   I = I + 1; #next edge#
IF Status EQ 'violation'
THEN #obtain the times of violation#
   FOR J = 1 TO 2; #compute the intersect points' time coordinates#
   Int_Pt(J).T = (SEGMENT.first_t - SEGMENT.second_t) *
         (DIST(SEGMENT.hz_first_vtx, SEGMENT.hz_sec_vtx) /
         DIST(SEGMENT.hz_first_vtx, Int_Pt(J).Hz_Comp)) +
         SEGMENT.first_t;
   Time_Viol_Start = MAX(Time_Overlap_Min, MIN(Int_Pt(1).T, Int_
      Pt(2).T);
   Time_Viol_End = MIN(Time_Overlap_Max, MAX(Int_Pt(1).T, Int_
      Pt(2).T);
   IF Time_Viol_Start EQ Time_Viol_End
   THEN #no violation as they intersect in a point#
      Status = 'no violation';
END Segment_Envelope_Intersect_Check;
```

FIGURE 4-36
SEGMENT_ENVELOPE_INTERSECT_CHECK (Concluded)

4.3.5 Encounter List Builder

4.3.5.1 Mission

The mission of the Encounter List Builder is to insert into the global table, ENCOUNTERS, information describing the violation detected by the three previous components (Time Check, Altitude Check, and Horizontal Check) of the Fine Filter. Given that a complete encounter may extend over several segments for one or both aircraft, this information may describe only part of an encounter. Under such circumstances, the information is merged with data associated with other portions of the encounter, if the data is already in the table as a result of previous iterations of the Fine Filter.

The ENCOUNTERS table is a source of encounter information available for use in the display provided to the controller.

4.3.5.2 Design Considerations and Component Environment

Input

The list of inputs to the Encounter List Builder consists of all those local tables and parameters which are used to describe the violation. They include the subject and nominee aircraft segments (SUBJECT_SEGMENT and NOMINEE_SEGMENT), unique identifiers for the subject and nominee aircraft flight plans (Subject_Fl_Id and Nominee_Fl_Id), advisory violation start and end times (Advisory_Time_Viol_Start and Advisory_Time_Viol_End), priority violation start and end times (Priority_Time_Viol_Start and Priority_Time_Viol_End), time of minimum separation between the aircraft in the horizontal plane (Time_Msep), and minimum separation distance in the plane (Msep_Dist). In addition, the Encounter List Builder accesses the global table ENCOUNTERS which it updates with information from the current encounter or portion of an encounter.

Output

The output of the Encounter List Builder is an updated version of the ENCOUNTERS Table.

4.3.5.3 Component Design Logic

The Encounter List Builder is called by the Fine Filter whenever the subject-nominee segment pair under consideration passes through the checks in the Time Check, Altitude Check, and Horizontal Check elements. It, in turn, calls the element

Violation Boundaries, which calculates the spatial coordinates of the two aircraft at the start and end of the advisory violation, and the elements Prefix Merge and Suffix Merge, which essentially merge the data associated with the current violation with that already in the table if the current violation is found to be part of an already identified encounter.

As indicated in Figure 4-37, which shows a PDL representation of the Encounter List Builder algorithm, Violation Boundaries is called first. The algorithm then iterates through each record in the global table ENCOUNTERS, searching for the possible occurrence of an encounter portion that immediately precedes or follows the current violation. Specifically, if the advisory violation end time of a record in the table is equal to the advisory violation start time of the current violation, then the record defines an encounter portion which immediately precedes and adjoins the current violation. Under such a circumstance, the Encounter List Builder Algorithm calls Prefix Merge which compares the data in the record with that of the current violation and changes the values of the violation parameters so they are descriptive of the merged encounter portions.

Similarly, if the advisory violation start time of the ENCOUNTERS record is equal to the advisory violation end time of the current violation, then the record describes an encounter portion which immediately follows the current violation. Suffix Merge is called to re-evaluate the violation parameters so that the parameters describe the merged encounter portions.

Subsequent to each merge, the corresponding record in the ENCOUNTERS Table is deleted. Once all of the records in the table are tested, the new violation parameters are inserted in the table in the form of a new record. If the table does not contain portions of the same encounter as the current violation, the algorithm simply inserts the unaltered values of the violation parameters (as calculated by the Horizontal Check element) directly into the table.

It should be noted that if the table contains an encounter portion that precedes the current violation and another portion that follows it, the algorithm will combine all three portions, and the data inserted into the table will describe the new encounter or encounter portion. In the following paragraphs, the phrase "current violation" is used to refer to either the violation identified by the Fine Filter in this iteration or

```
ROUTINE Encounter_List_Builder;
PARAMETERS SUBJECT_SEGMENT IN, NOMINEE_SEGMENT IN, Subject_Fl_Id IN,
   Nominee_Fl_Id IN, Advisory_Time_Viol_Start IN, Advisory_Time_
   Viol_End IN, Priority_Time_Viol_Start IN, Priority_Time_Viol_End
   IN, Time_Msep IN, Msep_Dist IN;
REFER TO GLOBAL Advisory_Sept IN, Priority_Sept IN, ENCOUNTERS INOUT;
DEFINED IN GLOSSARY
   Subject_Fl_Id
   Nominee_Fl_Id
   Advisory_Time_Viol_Start
   Advisory_Time_Viol_End
   Priority_Time_Viol_Start
   Priority_Time_Viol_End
   Time_Msep
   Msep_Dist
   Subject_Viol_Start_Pt
   Subject_Viol_End_Pt
   Nominee_Viol_Start_Pt
   Nominee_Viol_End_Pt
   SUBJECT_SEGMENT
   NOMINEE_SEGMENT;
```

FIGURE 4-37
ENCOUNTER LIST BUILDER

```
# calculate the spatial coordinates of the subject and nominee #
# aircraft at the start and end of the violation                #
CALL Violation_Boundaries (SUBJECT_SEGMENT IN, NOMINEE_SEGMENT
   IN, Advisory_Time_Viol_Start IN, Advisory_Time_Viol_End IN,
   Subject_Viol_Start_Pt OUT, Subject_Viol_End_Pt OUT, Nominee_
   Viol_Start_Pt OUT, Nominee_Viol_End_Pt OUT);
REPEAT FOR EACH ENCOUNTERS RECORD
   WHERE ENCOUNTERS.first_fl_id EQ Subject_Fl_Id AND ENCOUNTERS.
      second_fl_id EQ Nominee_Fl_Id;
   # if the data in this record describe an encounter portion which #
   # adjoins the newly detected encounter, re-evaluate the violation#
   # parameters so that they describe the combined encounter #
   IF ENCOUNTERS.adv_viol_end_time EQ Advisory_Time_Viol_Start
   THEN
      CALL Prefix_Merge (ENCOUNTERS IN, Advisory_Time_Viol_Start
         INOUT, Priority_Time_Viol_Start INOUT, Priority_Time_Viol_
         End INOUT, Msep_Dist INOUT, Time_Msep INOUT, Subject_Viol_
         Start_Pt INOUT, Nominee_Viol_Start_Pt INOUT);
      DELETE FROM ENCOUNTERS; #Remove current record being#
         #considered#
   ELSE
      IF ENCOUNTERS.adv_viol_start_time EQ Advisory_Time_Viol_End
      THEN
         CALL Suffix_Merge (ENCOUNTERS IN, Advisory_Time_Viol_End
            INOUT, Priority_Time_Viol_Start INOUT, Priority_Time_
            Viol_End INOUT, Msep_Dist INOUT, Time_Msep INOUT,
            Subject_Viol_End_Pt INOUT, Nominee_Viol_End_Pt INOUT);
         DELETE FROM ENCOUNTERS;
# record the encounter data #
INSERT INTO ENCOUNTERS (first_fl_id = Subject_Fl_Id, second_fl_id =
   Nominee_Fl_Id, adv_viol_start_time = Advisory_Time_Viol_Start,
   adv_viol_end_time = Advisory_Time_Violation_End, display_as
   advisory_time = Advisory_Time_Viol_Start - Advisory_Sept, prior
   viol_start_time = Priority_Time_Viol_Start, prior_viol_end_time =
   Priority_Time_Violation_End, display_as_priority_time = Priority
   Time_Viol_Start - Priority_Sept, msep_time = Time_Msep, msep
   distance = Msep_Dist, f11_viol_start_pt = Subject_Viol_Start_Pt,
   f11_viol_end_pt = Subject_Viol_End_Pt, f12_viol_start_pt =
   Nominee_Viol_Start_Pt, f12_viol_end_pt = Nominee_Viol_End_Pt);
END Encounter_List_Builder;
```

FIGURE 4-37
ENCOUNTER LIST BUILDER (Concluded)

Figure 4-39 contains a PDL presentation of the Prefix Merge Algorithm.

Suffix Merge

This element provides values for the parameters of the complete encounter or encounter portion that results from the merging of the current violation and the encounter portion which follows it in time. The data for the encounter portion which follows it in time is contained in the single record table called SUFFIX. The general nature of the Suffix Merge algorithm is the same as that of Prefix Merge. Only the list of parameters that are reset is different.

Figure 4-40 contains a PDL presentation of the algorithm.

4.4 Maintenance

The role of the Maintenance subfunction is to maintain updated versions of the various global and shared local tables used by FPCP. Whenever the stimulus that invokes FPCP indicates a trajectory update, and the nature of this update is a revised flight plan, an outbound flight or a terminated flight, all references to the flight identification are removed in the shared local tables SPARSE_TREE, ALLOBJECT_ BLOCKS, ALLOBJECT_TREE, and in the global tables SPARSE_CELLS and ENCOUNTERS. The removal of the references to the flight identification from all but the ALLOBJECT_BLOCKS and the ALLOBJECT_TREE tables is a trivial process involving the deletion of appropriate records from these tables. Hence, the process is not discussed further here. On the other hand, removing the references to the flight identification from the ALLOBJECT_BLOCKS and ALLOBJECT_TREE tables is not trivial. The routine that accomplishes this task, Delete Aircraft, is considered to be a principal component of the Maintenance subfunction.

After the data associated with an aircraft with a revised trajectory have been removed from the appropriate tables, the various Coarse Filter and Fine Filter tests are invoked. These same tests are invoked directly (without any prior updating of the tables by Maintenance) for a new trajectory, a horizon update, and a trial probe. In the case of a trajectory update (new or revised) or a horizon update, onc he tests are completed, the subject aircraft is referenced as an object aircraft in the tables. Thus, information about its trajectory gets included in the ALLOBJECT_TREE table. The ALLOBJECT_BLOCKS Table is modified to reflect the new occupancy

```
ROUTINE Prefix_Merge;
PARAMETERS PREFIX IN, Advisory_Time_Viol_Start INOUT, Priority_Time_
   Viol_Start INOUT, Priority_Time_Viol_End INOUT, Msep_Dist INOUT,
   Time_Msep INOUT, Subject_Viol_Start_Pt INOUT, Nominee_Viol_Start_
   Pt INOUT;
DEFINED IN GLOSSARY
   Advisory_Time_Viol_Start
   Priority_Time_Viol_Start
   Priority_Time_Viol_End
   Msep_Dist
   Time_Msep
   Subject_Viol_Start_Pt
   Nominee_Viol_Start_Pt;
DEFINE TABLES
   PREFIX            Single record table which contains data
                        describing an encounter portion which
                        immediately precedes the current encounter
                        portion in real time; fields defined like
                        global table ENCOUNTERS;
Advisory_Time_Viol_Start = PREFIX.adv_viol_start_time;
IF PREFIX.prior_viol_start_time NE NULL
THEN
   Priority_Time_Viol_Start = PREFIX.prior_viol_start_time;
   IF Priority_Time_Viol_End EQ NULL
   THEN
      Priority_Time_Viol_End = PREFIX.prior_viol_end_time;
IF PREFIX.msep_distance NE NULL  AND  Msep_Dist NE NULL
THEN
   IF PREFIX.msep_distance LT Msep_Dist
   THEN
      Msep_Dist = PREFIX.msep_distance;
      Time_Msep = PREFIX.msep_time;
ELSE
   IF Msep_Dist EQ NULL
   THEN
      Msep_Dist = PREFIX.msep_distance;
      Time_Msep = PREFIX.msep_time;
Subject_Viol_Start_Pt = PREFIX.f11_viol_start_pt;
Nominee_Viol_Start_Pt = PREFIX.f12_viol_start_pt;
END Prefix_Merge;
```

FIGURE 4-39
PREFIX MERGE

```
ROUTINE Suffix_Merge;
PARAMETERS SUFFIX IN, Advisory_Time_Viol_End INOUT, Priority_Time_
   Viol_Start INOUT, Priority_Time_Viol_End INOUT, Msep_Dist INOUT,
   Time_Msep INOUT, Subject_Viol_End_Pt INOUT, Nominee_Viol_End_Pt
   INOUT;
DEFINED IN GLOSSARY
   Advisory_Time_Viol_End
   Priority_Time_Viol_Start
   Priority_Time_Viol_End
   Msep_Dist
   Time_Msep
   Subject_Viol_End_Pt
   Nominee_Viol_End_Pt;
DEFINE TABLES
   SUFFIX              Single record table which contains data
                          describing an encounter portion which
                          immediately succeeds the current encounter
                          portion in real time; fields defined like
                          global table ENCOUNTERS;
Advisory_Time_Viol_End = SUFFIX.adv_viol_end_time;
IF Priority_Time_Viol_End EQ NULL
THEN
   Priority_Time_Viol_Start = SUFFIX.prior_viol_start_time;
   Priority_Time_Viol_End = SUFFIX.prior_viol_end_time;
ELSE
   IF SUFFIX.prior_viol_end_time NE NULL
   THEN
      Priority_Time_Viol_End = SUFFIX.prior_viol_end_time;
IF SUFFIX.msep_distance NE NULL  AND  Msep_Dist NE NULL
THEN
   IF SUFFIX.msep_distance LT Msep_Dist
   THEN
      Msep_Dist = SUFFIX.msep_distance;
      Time_Msep = SUFFIX.msep_time;
ELSE
   IF Msep_Dist EQ NULL
   THEN
      Msep_Dist = SUFFIX.msep_distance;
      Time_Msep = SUFFIX.msep_time;
Subject_Viol_End_Pt = SUFFIX.f11_viol_end_pt;
Nominee_Viol_End_Pt = SUFFIX.f12_viol_end_pt;
END Suffix_Merge;
```

FIGURE 4-40
SUFFIX MERGE

count of each block. This is all accomplished by Insert Aircraft, the second of Maintenance's two components.

For the case of the trial probe, the ALLOBJECT_TREE and ALLOBJECT_ BLOCKS tables are not altered. This is to prevent a new subject trajectory from being declared in conflict with the trial trajectory while a controller decides whether to accept a trial flight plan or not. In either case, whether a trial trajectory is accepted or not, all references to the trial flight identification are removed from SPARSE_CELLS, SPARSE_TREE, and ENCOUNTERS once the controller's decision is entered into the automation system.

Figure 4-41 illustrates the organizational structure of the Maintenance subfunction.

4.4.1 Delete Aircraft

4.4.1.1 Mission

The purpose of the Delete Aircraft component is to remove obsolete data in ALLOBJECT_TREE and ALLOBJECT_BLOCKS pertaining to the aircraft under consideration by the Maintenance subfunction. This occurs every time that a trajectory is revised due to resynchronization or a request for a flight plan change. The intent is to "clean the slate" before invoking the Coarse Filter, thus making it possible for the Coarse and Fine Filters to treat this aircraft as if it were a new subject aircraft entering the planning region.

4.4.1.2 Design Considerations and Component Environment

The Delete Aircraft routine is invoked by the Maintenance subfunction whenever the trajectory of an aircraft in the planning region is revised. A discussion of the tree traversal technique used in this algorithm is given in Appendix C.

Input

The inputs to the Delete Aircraft component consist of a combination of global tables, shared local tables and variables, and input parameters. The input parameters to the routine are Current_Node_Id, which identifies the node currently being traversed in a search of the subtree to be deleted, and Level, which specifies the node's level. Shared local data consist of the SPARSE_TREE and ALLOBJECT_BLOCK tables and the variable Max_Level. SPARSE_TREE defines the structure of the tree associated with the subject aircraft.

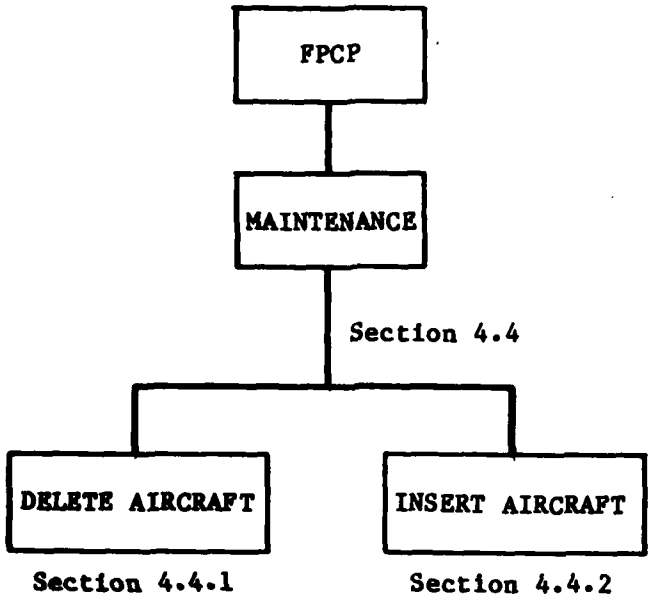


FIGURE 4-41
MAINTENANCE ORGANIZATIONAL STRUCTURE

The table includes data for other aircraft as well; the specific subject being referenced is keyed on the global variable Subject_Fl_Id. ALLOBJECT_BLOCKS contains the occupancy count (that is, the number of aircraft in a given block) of the current root and is keyed on Current_Node_Id. The shared local parameter Max_Level specifies the cell (leaf) level of the tree. Finally, the global table ALLOBJECT_TREE, which is both input to and updated by the element Delete Subtree, defines each node and all its children in the subject's tree.

Output

The Delete_Aircraft and Delete_Subtree routines update both ALLOBJECT_TREE and ALLOBJECT_BLOCKS by deleting the appropriate records from each table as described below.

4.4.1.3 Component Design Logic

The Delete Aircraft algorithm is recursive. The algorithm performs a preorder traversal through the tree searching for a subtree to delete. Once found, a post order traversal of that subtree is performed to delete each node of that subtree. Each invocation corresponds to a move down the octal tree modelled by the table ALLOBJECT_TREE. Essentially, Delete Aircraft searches those nodes (or, equivalently, blocks of the (x,y,t) Coarse Filter grid) in ALLOBJECT_TREE which are occupied by the subject aircraft. If the subject aircraft is the sole occupant of the input node (or, more accurately, of its corresponding block), the entire subtree (whose root is the input note) is deleted. Deletions of such subtrees are performed by the element Delete_Subtree (which is itself recursive). If the subject aircraft is not the sole occupant, the algorithm reduces the current value of Occupancy_Count in the table ALLOBJECT_BLOCKS associated with the node by one. It then loops on the node's children, calling itself whenever one of the child nodes is occupied by the subject aircraft.

Figure 4-42 shows a PDL representation of the Delete Aircraft algorithm.

Delete Subtree

The Delete Subtree element performs a postorder traversal of the ALLOBJECT_TREE starting at the node whose Occupancy_Count is 1. First, the ALLOBJECT_BLOCK record involving the root node is deleted. Then Delete Subtree calls itself for a given child of the root node and recursively traverses the tree until

```
ROUTINE Delete_Aircraft;
PARAMETERS Subject_Fl_Id IN, Current_Node_In IN, Level IN;
REFER TO SHARED LOCAL SPARSE_TREE IN, Max_Level IN, ALLOBJECT_BLOCKS
   INOUT;
DEFINE VARIABLES
   Subject_Fl_Id      Unique identifier for the flight plan of the
                         subject aircraft
   Current_Node_Id    Identifier of the current root of the subtree
                         being traversed
   Level              Current level of the root of the subtree being
                         traversed
   Occupancy_Count    Number of aircraft co-occupying the block;
IF Level LT Max_Level
THEN #block level#
   #test to see if the subject is the only aircraft in block#
      # of the ALLOBJECT tree#
   SELECT FIELDS occupancy_count
      INTO Occupancy_Count
      FROM ALLOBJECT_BLOCKS
      WHERE ALLOBJECT_BLOCKS.node_id EQ Current_Node_Id;
   IF Occupancy_Count EQ 1 #last one#
   THEN #delete the subtree with root equal to Current_Node_Id#
      CALL Delete_Subtree(Current_Node_Id IN, Level IN);
   ELSE #more than one aircraft in the block#
      #reduce number of aircraft in the block#
      UPDATE IN ALLOBJECT_BLOCKS
         (occupancy_count = occupancy_count - 1)
         WHERE ALLOBJECT_BLOCKS.node_id EQ Current_Node_Id;
      #compare children of the SPARSE tree and the ALLOBJECT tree#
         #to see which subtree if any should be deleted#
      REPEAT FOR EACH SPARSE_TREE RECORD #for each child#
         #the SPARSE tree is identified by the flight plan id and#
            #the children are identified by all records with the#
            #same node_id (e.g. Current_Node_Id)#
         WHERE SPARSE_TREE.Fl_Id EQ Subject_Fl_Id AND
            SPARSE_TREE.node_id EQ Current_Node_Id;
         #recursively check to see if the subtree with root node_id#
            #equal to SPARSE_TREE.child_id should be deleted#
         CALL Delete_Aircraft(SPARSE_TREE.child_id IN, Level+1 IN);
END Delete_Aircraft;
```

FIGURE 4-42
DELETE_AIRCRAFT

the leaf level is reached. Once the leaf is reached, this routine returns to the previous level and deletes the ALLOBJECT_TREE record associated with the root at that level. It then invokes itself for the next child, continuing until all children have been processed and then returns to the previous level and repeats the process.

Figure 4-43 shows a PDL representation of the Delete Subtree algorithm.

4.4.2 Insert Aircraft

4.4.2.1 Mission

The mission of the Insert Aircraft component of Maintenance is to modify the ALLOBJECT_TREE and ALLOBJECT_BLOCKS in order that they include the trajectory data of the subject aircraft. This is accomplished by combining (obtaining the union of) the ALLOBJECT_TREE and the SPARSE_TREE records associated with the subject aircraft.

4.4.2.2 Design Considerations and Component Environment

The Insert Aircraft component is invoked by the Maintenance subfunction after a new or revised subject aircraft trajectory has been processed by the Coarse and Fine Filters. A discussion of the tree traversal techniques used by this algorithm is given in Appendix C.

Input

The inputs to the Insert Aircraft component are basically the same global tables, local shared tables and variables, and input parameters that are required by the Delete Aircraft component (Section 4.4.1, Delete Aircraft). In Delete Aircraft, the SPARSE_TREE records refer to the tree prior to any trajectory revisions, whereas in Insert Aircraft, they refer instead to the tree after the revisions have been made.

Output

Like Delete Aircraft, Insert Aircraft is a recursive algorithm. Every invocation results in a modification of the ALLOBJECT_TREE and associated data in ALLOBJECT_BLOCKS, which are the outputs of this routine. When the process is finished, the output is a new ALLOBJECT_TREE containing references to the trajectory data for the subject aircraft.

```
ROUTINE Delete_Subtree;
PARAMETERS Current_Node_Id IN, Level IN;
REFER TO SHARED LOCAL ALLOBJECT_TREE INOUT, Max_Level IN,
   ALLOBJECT_BLOCKS INOUT;
DEFINE VARIABLES
   Current_Node_Id   Identifier of the current root of the subtree
                        being traversed
   Level             Current level of the root of the subtree being
                        traversed;
IF Level LT Max_Level
THEN
   DELETE FROM ALLOBJECT_BLOCKS #delete records corresponding to the#
      #tree that will be deleted#
      WHERE ALLOBJECT_BLOCKS.node_id EQ Current_Node_Id;
   #delete subtree nodes#
   REPEAT FOR EACH ALLOBJECT_TREE RECORD #for each child#
      #the children are identifed by all records with the same node_#
         #id#
      WHERE ALLOBJECT_TREE.node_id EQ Current_Node_Id;
      #delete next level of the current subtree referenced by the#
         #child#
      CALL Delete_Subtree(ALLOBJECT_TREE.child_id IN, Level+1 IN);
      DELETE FROM ALLOBJECT_TREE; #delete current child record;#
         #the record to delete is known from the above WHERE clause#
ELSE; #nothing to delete at leaf level, since the leaf node is#
         #referred to by the last non-leaf node.  This reference is#
         #to SPARSE_CELLS#
END Delete_Subtree;
```

FIGURE 4-43
DELETE_SUBTREE

4.4.2.3 Component Design Logic

The Insert Aircraft algorithm is a recursive procedure. Each invocation of the algorithm corresponds to a move one level down the ALLOBJECT_TREE. In essence, the algorithm copies all those nodes which are in the SPARSE_TREE associated with the subject aircraft, but not in the ALLOBJECT_TREE, and attaches them in the proper positions of the ALLOBJECT_TREE.

The algorithm begins by adding one to the current value of the occupancy count in the ALLOBJECT_BLOCK associated with the root (that is, key on Current_Node_Id). This is to indicate that a new aircraft has been added to the list of object aircraft in the ALLOBJECT_TREE. The algorithm then loops on the children of the SPARSE_TREE and ALLOBJECT_TREE roots in parallel (that is, all records with node_id equal to the Current_Node_Id). If it finds that a child node exists in the SPARSE_TREE, but not in the ALLOBJECT_TREE, it adds records to the ALLOBJECT_TREE associated with the current root and all children found in SPARSE_TREE. The outcome is the creation of a new version of the ALLOBJECT_TREE which contains a new node indicating occupancy of a block that was previously unoccupied. Following this procedure, the algorithm invokes itself, replacing the previous root by the child node it has just created in ALLOBJECT_TREE and the parallel child node in the SPARSE_TREE. It increments the value of Level by one, signifying that the following iteration will focus on the next level down in the two trees. This process will be repeated until the last level of both trees is reached.

In addition to invoking itself whenever a node is copied, the algorithm also invokes itself if it discovers that a child node exists in both trees. The new roots become the node identifiers associated with the two child nodes and the value of Level is incremented by one.

During each iteration of the algorithm, Level is compared to Max_Level. If it is determined that Max_Level has been reached, the algorithm returns to the previous level and repeats the process for the next child at that level.
Figure 4-44 shows a PDL version of the Insert Aircraft algorithm.

```
ROUTINE Insert_Aircraft;
PARAMETERS Subject_Fl_Id IN, Current_Node_Id IN, Level IN;
REFER TO SHARED LOCAL SPARSE_TREE IN, ALLOBJECT_TREE INOUT, Max_
   Level IN, ALLOBJECT_BLOCKS INOUT;
DEFINE VARIABLES
   Subject_Fl_Id      Unique identifier for the flight plan of the
                         subject aircraft
   Current_Node_Id    Identifier of the root of the current subtree
                         being traversed
   Level              Level of the root of the current subtree;
IF Level LT Max_Level
THEN
   #check if block already exists#
   IF COUNT(ALLOBJECT_BLOCKS.node_id EQ  Current_Node_Id) EQ 0
   THEN #new block, add it#
      INSERT INTO ALLOBJECT_BLOCKS (node_id = Current_Node_Id,
         occupancy_count = 1);
   ELSE #block already exists, update count#
      UPDATE IN ALLOBJECT_BLOCKS(occupancy_count = occupancy_count +
         1)
         WHERE ALLOBJECT_BLOCKS.node_id EQ Current_Node_Id;
   REPEAT FOR EACH SPARSE_TREE RECORD #for each child#
      #the SPARSE tree is identified by the flight plan id. and the#
         #children are identified as all records with the same#
         #node_id (that is, Current_Node_Id)#
      WHERE SPARSE_TREE.node_id EQ Current_Node_Id AND SPARSE_
         TREE.fl_id EQ Subject_Fl_Id;
      #check for matching child block in the ALLOBJECT tree#
      IF COUNT(ALLOBJECT_TREE.node_id EQ Current_Node_Id AND
         ALLOBJECT_TREE.child_id EQ SPARSE_TREE.child_id) EQ 0
      THEN #child block not found#
         #add child to ALLOBJECT tree#
         INSERT INTO ALLOBJECT_TREE (node_id = Current_Node_Id,
            child_id = SPARSE_TREE.child_id);
      #check next level for insertion#
      CALL Insert_Aircraft(SPARSE_TREE.child_id IN, Level + 1 IN);
ELSE; #nothing at leaf to create, since the leaf node data is#
         #referred to by the child of last nonleaf node.  The#
         #leaf level data is in SPARSE_CELLS#
END Insert_Aircraft;
```

FIGURE 4-44
INSERT AIRCRAFT

APPENDIX A

FLIGHT PLAN CONFLICT PROBE DATA

SPARSE_TREE:

FL_ID	NODE_ID	CHILD_ID

This table defines the blocks of the airspace grid through which each flight plan trajectory passes and their relationships to larger blocks in the grid.

FL_ID	Unique identifier which distinguishes one flight plan from all other flight plans currently defined on the system
NODE_ID	Unique identifier of a block of airspace in an x,y,t grid
CHILD_ID	Unique identifier of a block of airspace in an x,y,t grid which is an octant of that given by NODE_ID

BUFFER_CELLS:

NODE_ID	min_z	max_z	entry_time	exit_time

This table defines the cells in the vicinity of the flight plan trajectory of the subject aircraft, the range of altitudes the trajectory covers in each cell, and the times associated with the cusp preceding entry and the cusp following exit for each cell.

NODE_ID	Unique identifier of an airspace cell in an x,y,t grid
min_z	The lowest altitude through which the subject aircraft's trajectory passes in the vicinity of this cell
max_z	The highest altitude through which the subject aircraft's trajectory passes in the vicinity of this cell
entry_time	The time associated with the cusp which precedes entry into the vicinity of this cell
exit_time	The time associated with the cusp which follows exit from the vicinity of this cell

("Vicinity" means the cell and its orthogonal and diagonal neighbors.)

BUFFER_TREE:

NODE_ID	CHILD_ID

This table defines the blocks of the airspace grid in the vicinity of the flight plan trajectory of the subject aircraft and their relationships to larger blocks in the grid.

NODE_ID Unique identifier of a block of airspace in an x,y,t grid

CHILD_ID Unique identifier of a block of airspace in an x,y,t grid which is an octant of that given by NODE_ID

ALLOBJECT_BLOCKS:

NODE_ID	occupancy_count

This table defines the blocks of the airspace grid through which any current flight plan trajectory passes, the number of octants in each block through which the trajectories pass, and the number of trajectories which pass through each block.

NODE_ID Unique identifier of a block of airspace in an x,y,t grid

occupancy_count The number of trajectories which pass through this block

ALLOBJECT_TREE:

NODE_ID	CHILD_ID

This table defines the blocks of the airspace grid through which any current flight plan trajectory passes and their relationships to larger blocks in the grid.

NODE_ID Unique identifier of a block of airspace in an x,y,t grid

CHILD_ID Unique identifier of a block of airspace in an x,y,t grid which is an octant of that given by NODE_ID

NOMINEES:

FL_ID	NODE_ID	subject_entry_time	subject_exit_time	nominee_entry_time	nominee_exit_time

This table defines the cells for which other flight plan trajectories may be in conflict with the subject aircraft's trajectory, the times associated with the cusp preceding entry and the cusp following exit for each cell for both the subject aircraft's trajectory and the nominee aircraft's trajectory.

FL_ID	Unique identifier which distinguishes one nominee flight plan from all other flight plans currently defined on the system
NODE_ID	Unique identifier of an airspace cell in an x,y,t grid
subject_entry_time	The time associated with the cusp which precedes entry into this cell for the subject aircraft's trajectory
subject_exit_time	The time associated with the cusp which follows exit from this cell for the subject aircraft's trajectory
nominee_entry_time	The time associated with the cusp which precedes entry into this cell for the nominee aircraft's trajectory
nominee_exit_time	The time associated with the cusp which follows exit from this cell for the nominee aircraft's trajectory

SPARSE_CELLS:

FLIGHT_PLAN_ID	NODE_ID	min_z	max_z	entry_time	exit_time

This table defines the cells which each flight plan trajectory enters, the range of altitudes the trajectory covers in each cell, and the times associated with the cusp preceding entry and the cusp following exit for each cell.

FLIGHT_PLAN_ID	Unique identifier which distinguishes one flight plan from all other flight plans currently defined on the system
NODE_ID	Unique identifier of an airspace cell in an x,y,t grid
min_z	The lowest altitude through which this flight plan trajectory passes in this cell
max_z	The highest altitude through which this flight plan trajectory passes in this cell
entry_time	The time associated with the cusp which precedes entry into this cell
exit_time	The time associated with the cusp which follows exit from this cell

SHARED LOCAL VARIABLES

H_Cell_Dimension	Quantization size for cells in horizontal x-y
Max_Level	The level number associated with the leaf level of the octal trees SPARSE_TREE, BUFFER_TREE and ALLOBJECT_TREE
Real_Subject_Fl_Id	In the case of a trial probe, this variable contains the flight plan identifier associated with the subject aircraft's actual flight plan and Subject_Fl_Id contains a dummy flight plan identifier associated with the trial flight plan
T_Cell_Dimension	Quantization size for cells in time
T_Offset	A translation in t in the conversion of time to cell coordinates
Trial_Flag	A flag indicating whether or not the FPCP was called for a trial probe
X_Offset	A translation in y in the conversion of geometric to cell coordinates
Y_Offset	A translation in x in the conversion of geometric to cell coordinates

APPENDIX B

MATHEMATICAL DERIVATION OF FORMULAS

B.1 The Time of Violation Formulas

Definitions: $D(t)$ = horizontal separation distance between the two aircraft at time t

T = Time_Overlap_Min

$P_s(t)$ = position vector of subject aircraft at time t

$P_n(t)$ = position vector of nominee at time t

V_s = velocity vector of subject aircraft (assumed to be constant over the length of the segment)

V_n = velocity vector of nominee (also assumed to be constant)

$P_r(t) = P_s(t) - P_n(t)$ = relative position vector at time t

$V_r = V_s - V_n$ = relative velocity vector

$\| \ \|$ = norm (length) of a vector

$\cdot$ = dot product of two vectors

Analysis:

First calculate D(t):

$$D(t) = \|P_s(t) - P_n(t)\| = \|P_r(t)\| = [P_r(t) \cdot P_r(t)]^{\frac{1}{2}}$$

$$= \left\{[P_r(T) + (t - T)\, V_r] \cdot [P_r(T) + (t-T)\, V_r]\right\}^{\frac{1}{2}}$$

$$= \left\{ \|P_r(T)\|^2 + 2(t-T)P_r(T)\cdot V_r + (t-T)^2 \|V_r\|^2 \right\}^{\frac{1}{2}}$$

To calculate the time at which D(t) is equal to Seph, i.e., the starting and ending times of the violation, set $D(t) = Seph$ or, equivalently, $D^2(t) = Seph^2$ (to produce a quadratic equation) and solve for t.

Let

$$A = \|V_r\|^2$$

$$B = 2P_r(T) \cdot V_r$$

$$C = \|P_r(T)\|^2;$$

then the above equation becomes:

$$A(t-T)^2 + B(t-T) + C - Seph^2 = 0$$

If the discriminant of this equation, $B^2 - 4A(C - Seph^2)$ is less than 0, then the roots are imaginary, implying that the separation distance is never equal to Seph; consequently no encounter is predicted. If the discriminant is 0, there are two real and equal roots, meaning that the separation distance is equal to Seph at some time t*, but is never less than Seph which would be considered a violation of the FPCP horizontal separation standard. Thus, in this case, as in the case above, no encounter is predicted. If the discriminant is greater than 0, then there are two real and unequal roots which provide the starting and ending times of a violation. These times are obtained by way of the quadratic formula.

$$Time\_Viol\_Start = \frac{-B - [B^2 - 4A(C - Seph^2)]^{\frac{1}{2}}}{2A} + T$$

$$Time\_Viol\_End = \frac{-B + [B^2 - 4A(C - Seph^2)]^{\frac{1}{2}}}{2A} + T$$

B.2 The Minimum Separation Formulas

Definitions: Same as those above

Analysis: First calculate D(t) as above, obtaining

$$D(t) = \left\{\|P_r(T)\|^2 + 2(t-T)P_r(T) \cdot V_r + (t-T)^2 \|V_r\|^2\right\}^{\frac{1}{2}}$$

$$= \left\{A(t-T)^2 + B(t-T) + C\right\}^{\frac{1}{2}}$$

where

$$A = \| V_r \|^2$$

$$B = 2P_r(T) \cdot V_r$$

$$C = \| P_r(T) \|^2;$$

To find the time t* when this function is a minimum (t* = Time_Msep), one needs to differentiate D(t), set the result equal to 0, and solve for t.

$$\frac{dD(t)}{dt} = \frac{2A(t-T) + B}{2\,[A(t-T)^2 + B(t-T) + C]^{\frac{1}{2}}} = 0$$

implying that

$$2A(t-T) + B = 0$$

and, consequently, that

$$t = t^* = \text{Time\_Msep} = \frac{-B}{2A} + T$$

Substituting into D(t) to obtain the minimum separation distance, one gets

$$\text{Msep\_Dist} = D(t^*) = [A(t^* - T)^2 + B(t^* - T) + C]^{\frac{1}{2}}$$

APPENDIX C

TREE TRAVERSAL TECHNIQUES USED BY THE COARSE FILTER AND MAINTENANCE

C.1 Recursion

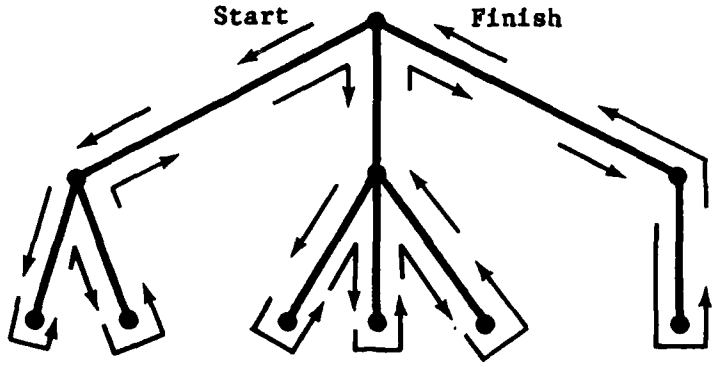
Data structures in the form of trees lead naturally to algorithms using recursion--that is, algorithms that call themselves as subroutines. Typically, a recursive algorithm must do some processing at each node of a tree. Some of this processing, which may be denoted P_{BEFORE}, may be required before any of the node's children are processed, while other processing, say P_{AFTER}, may be required only after all children have been processed. Other processing at the node may be required on a per-child basis; however, the bulk of this, especially that involving grandchildren and more remote descendants, is similar to P_{BEFORE} and P_{AFTER} one level down, with each child in turn taking the role of the parent.

A recursive algorithm generally looks only at a single node and its immediate children at any one time. In this volume, all trees have the property that all leaves are at the same level. An algorithm can then determine whether a node is a leaf by knowing its level. When a leaf is reached, there are no more children, and different processing, say P_{LEAF}, is performed. A typical recursive algorithm, which we denote Treesearch, is called via a statement such as CALL Treesearch (ROOT IN, 0 IN), while the procedure Treesearch looks like this:

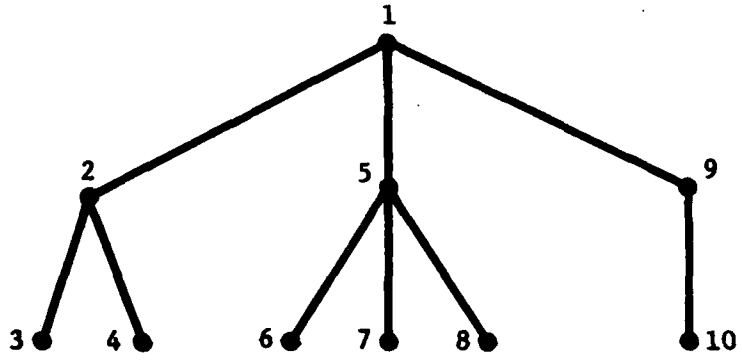
```
ROUTINE Treesearch (Node IN, Level IN);
IF Level EQ Leaf_Level
THEN
   CALL P_LEAF;
ELSE
   CALL P_BEFORE;
   REPEAT FOR EACH Child;
      CALL Treesearch (Child IN, Level + 1 IN);
   CALL P_AFTER;
END Treesearch;
```

Preorder and Postorder

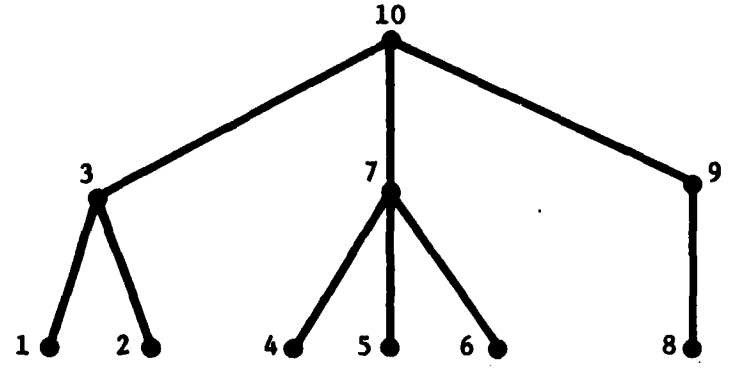
It is sometimes useful to assign an ordering to a tree's nodes. Consider an ant which starts at the root and crawls along the branches, always taking the leftmost unvisited branch, doubling back at the leaves, so that each branch is eventually traversed exactly once ending at the root, as shown



(a) Path of the ant around the tree



(b) Preorder. The ant assigns the labels as it visits the nodes



(c) Postorder. The ant withholds the parent's label until all its children are labeled

FIGURE C-1
ORDERINGS OF NODES ON A TREE

in Figure C-1a. If the ant counts (and labels) each node as it is first visited, the labels will appear as in Figure C-1b. Note that the root of each subtree has a lower label than any of its descendants. This ordering is called a preorder. Treesearch can be used as a template to generate a preordering of a tree: P_{BEFORE} and P_{LEAF} consist simply of labeling the current node with the next available number, starting with "1" at the root. P_{AFTER} is null. Specifically, using a global variable NUMBER initialized to "1" and an initial call such as CALL Preorder (ROOT IN, 0 IN), the algorithm is as follows:

```
ROUTINE Preorder (Node IN, LEVEL IN);
Assign Number to Node;
Number = Number + 1;
IF Level LT Leaf_Level
THEN
   REPEAT FOR EACH Child;
      CALL Preorder (Child IN, Level + 1 IN);
END Preorder;
```

Another useful ordering of a tree's nodes is the postorder, which resembles the preorder except that a node's label is withheld (by the ant) until all its children have received labels (Figure C-1c). Treesearch generates a postordering if the roles of P_{BEFORE} and P_{AFTER} ("label" and "do nothing") are reversed:

```
ROUTINE Postorder (Node IN, Level IN);
IF Level LT Leaf_Level
THEN
   REPEAT FOR EACH Child;
      CALL Postorder (Child IN, Level + 1 IN);
Assign Number to Node;
Number = Number + 1;
END Postorder;
```

C.2 Insertion To and Deletion From the ALLOBJECT Tree Using Treesearch

Treesearch can also be used as a template for inserting or deleting the cells of a grid chain (sparse or buffer) from the set of cells represented in the Allobject Tree. For an insert, the Allobject node corresponding to each subject node must be updated (to reflect one extra aircraft). Of course, if the Allobject Tree node does not exist, it must first be created, and created before any of its children can be added to the object tree. The insertion must be accomplished in preorder;

the node's creation belongs to P_{BEFORE}. For a delete, the Allobject node corresponding to each subject node must be updated (to reflect one fewer aircraft). If the count is thereby reduced to zero, the Allobject node must be deleted--but any of its children must be deleted first (otherwise they would become inaccessible). The deletion must be accomplished in postorder; the node's deletion belongs to P_{AFTER}. In an insert or delete, a leaf node may be treated like any other node, except that its children are not checked.

C.3 Coarse Filter Using Treesearch

Treesearch can serve as a template for the coarse filter itself. The search proceeds (in preorder) only along nodes found in both the subject and the Allobject Tree (no conflict can occur for a node unless both subject and object occupy the corresponding grid block). Thus, P_{BEFORE} consists of eliminating from further consideration any of a node's children not found in both trees. P_{LEAF} consists of adding the object to the table of nominees (after an altitude screening). P_{AFTER} is null:

```
ROUTINE Coarse_Filter (Node IN, Level IN);
IF Level LT Leaf_Level
THEN
   REPEAT FOR EACH of the eight possible children;
      IF Child exists in both trees
      THEN
         CALL Coarse_Filter (Child IN, Level + 1 IN);
ELSE
   IF Altitude conditions met
   THEN Add each object occupying this cell (corresponding
      to Node) to a table of nominees;
END Coarse_Filter;
```

Section 4.2 describes the details of an algorithm named Nominee Detection which is called by the Coarse Filter to perform this process.

APPENDIX D

GLOSSARY

Number in parentheses at the end of the definition refers to the section in which the term is first used.

AAS	Advanced Automation System (1.1)
Advisory message	Message displayed to the controller for conflicts not necessarily requiring prompt resolution; may be in the form of text and/or a graphic display (2.1.11)
Advisory separation criteria	The criteria used by the Fine Filter in the horizontal and time dimensions to declare an advisory conflict (2.1.11)
Advisory Seph	The horizontal distance used by FPCP for declaring an advisory conflict between two aircraft (2.1.11)
Advisory Sept	The time in advance of the advisory time of violation that is allowed to the controller to resolve the conflict (2.1.11)
Advisory time of violation	The initial time at which the distance between two aircraft trajectories falls below the advisory SEPH (2.1.11)
AERA	Automated En Route Air Traffic Control (1.4.1.1)
Air traffic controller	Same as "Controller" (1.4.1)
Airspace grid	Grid dividing the horizontal dimensions of the planning region over time into discrete cells (2.1.6)
Allobject Tree	A tree which is the union of all individual object trees (2.1.8)
Altitude restriction	A directive from a controller to a pilot to be at, at or above, or at or below a given altitude by a given point along the flight path (1.4.1)
Ancestor (of a tree node)	Either a parent of the node, or a parent of an ancestor of the node (2.1.8)

Area	Second level division (see "Center," "Sector") of the Continental United States airspace. Controllers are specially trained for an area's airspace, a region bounded horizontally by a polygon and stretching vertically up to 60,000 feet (1.4.1)
Area supervisor	The first-line supervisor of an area (1.4.1)
ARTCC	Air Route Traffic Control Center (see "Center") (1.4.1)
ATC	Air Traffic Control (1.1)
Block	A subset of the (x,y,t) airspace grid associated with one tree node (2.1.8)
Buffer grid chain	The sparse grid chain plus all the cells which share at least one vertex with a cell in the sparse grid chain (2.1.7)
Buffer Subject Tree	The tree formed from the subject's buffer grid chain (2.1.8)
Cell	Individual parallelepipeds in (x,y,t) space within the airspace grid (2.1.6)
Center	Administrative headquarters and operational facility for control of a first level division (see "Area," "Sector") of the Continental United States airspace (there are currently 20 centers); controls a region bounded horizontally by a polygon and stretching vertically from the center floor to 60,000 feet (1.4.1)
Child (of a tree node)	A node sharing an edge with the given node and having a higher level than the given node (2.1.8)
Coarse Filter	An algorithm that compares the Buffer Subject Tree to the Allobject Tree, eliminating from further consideration those objects which do not share occupancy of at least one cell with the subject (2.1.9)
Component	Third level algorithmic unit in breakdown of AERA (see "Function," "Subfunction," "Element") (1.3)

Conflict	Displayed violation of the FPCP advisory separation criteria by one aircraft's trajectory with respect to another aircraft's trajectory (1.5.1)
Controller	In this document, an en route radar controller as defined in "Glossary of Common Terms in Air Traffic Control Operations" [13] (1.4.1)
Cusp	A regular cusp or a maneuver envelope cusp; a point in (x,y,z,t) space (2.1.4)
Delta horizon	Interval at which horizon updates are invoked (2.1.2)
Descendant (of a tree node)	Either a child of the node, or a child of a descendant of the node (2.1.8)
Display-as-advisory time	The time at which an advisory message is first displayed to the controller (2.1.11)
Display-as-priority time	The time at which a priority message is first displayed to the controller (2.1.11)
DP	Density Probe (1.4.2.2)
Element	Fourth level algorithmic unit in breakdown of AERA (see "Function," "Subfunction," "Component") (1.3)
ELOD	Enroute Sector Loading (1.4.1.2.1)
Encounter	Violation of the FPCP separation criteria found by the Fine Filter between the trajectories of the subject and a nominee (may be too far in the future to display as a conflict) (2.1.10)
Encounter aircraft	Nominee aircraft whose trajectory is in violation of the FPCP separation criteria relative to the subject's trajectory according to the Fine Filter (2.1.10)
FAA	Federal Aviation Administration (1.1)
Fine Filter	An algorithm that tests subject-nominee segment pairs against FPCP separation criteria using rigorous mathematical analyses (2.1.10)

Flight plan	Pilot's intended route to reach his destination as cleared by the air traffic control system (1.4.1)
FPCP	Flight Plan Conflict Probe (1.1)
FPCP advisory separation criteria	Same as "Advisory separation criteria" (2.1.11)
FPCP horizon update	Same as "horizon update" (2.1.2)
FPCP priority separation criteria	Same as "Priority separation criteria" (2.1.11)
FPCP trajectory update	Same as "trajectory update" (2.1.3)
Function	A major building block of AERA--a principal algorithm which is the top level unit in the breakdown of AERA (see "Subfunction," "Component," "Element") (1.1)
GCG	Grid Chain Generator (3.3.1)
Grid cell	Same as "Cell" (2.1.6)
Grid chain	List of an aircraft's occupied cells (2.1.7)
Hold	Same as "Holding pattern" (2.1.5)
Holding pattern	An aircraft maneuver to delay its en route progress; usually a circling or spiraling within a specified airspace (2.1.5)
Holding pattern cusp	The entry or exit point of the holding pattern expressed in spatial and temporal coordinates (2.1.5)
Holding pattern segment	Portion of the trajectory containing a holding pattern and defined by a pair of holding pattern cusps (2.1.5)
Horizon update	A periodic updating of the time horizon which causes an invocation of FPCP (2.1.2)

ID	Identification (1.5.1)
Independent Variable	The variable (x, y, or t) in which a trajectory segment changes most rapidly (4.1.1.3)
Leaf (of a tree)	A node with no children (2.1.8)
Level (of a tree node)	Nonnegative integer assigned to the node (number of edges on path to root)
Maneuver envelope	The geometric structure which encloses an airspace-sweeping maneuver (2.1.5)
Maneuver envelope cusp	The entry or exit point of the maneuver envelope expressed in spatial and temporal coordinates (2.1.5)
Maneuver envelope segment	Portion of the trajectory containing a maneuver envelope and defined by a pair of maneuver envelope cusps (2.1.5)
NAS	National Airspace System (1.1)
NASP	National Airspace System Plan (1.4.1)
Node	Same as "tree node" (2.1.8)
Nominee	An aircraft which was not eliminated from further consideration by the Coarse Filter and therefore will have its trajectory examined by the Fine Filter (2.1.9)
Nominee aircraft	Same as "Nominee" (2.1.9)
Object	An aircraft (which is not the subject) whose current trajectory has already been processed by FPCP (1.5.2)
Occupied cell	Cell selected by one of two mathematical formulas to be in the grid chain for a trajectory; there are sparse and buffer criteria for determining occupancy (2.1.7)
Occupied block	A block containing an occupied cell (2.1.8)

Octant (of a block)	One of eight blocks obtained by dividing the given block in half along each of the x, y, and t axes (2.1.8)
Parent (of a tree node)	A node sharing an edge with the given node and having a lower level than the given node (2.1.8)
PDL	Program design language (1.2)
Planning region	A center's airspace plus a buffer zone around it for handoffs between centers (1.5.2)
Postorder	Ordering of tree nodes by labelling child nodes before their parent node (3.3.2.1.1)
Preorder	Ordering of tree nodes by labelling a parent node before its child nodes (3.3.2.1.1)
Priority message	Message displayed to the controller when two trajectories are in violation of FPCP priority separation criteria (2.1.11)
Priority separation criteria	The criteria used by the Fine Filter in the horizontal and time dimensions to declare a priority conflict (2.1.11)
Priority Seph	The horizontal distance used by FPCP for declaring a priority conflict between two aircraft (2.1.11)
Priority Sept	The time in advance of the priority time of violation that is allowed to the controller to resolve the conflict (2.1.11)
Priority time of violation	The initial time at which the distance between two aircraft trajectories falls below the priority SEPH (1.5.2)
Regular cusp	One of the endpoints of a segment expressed in spatial and temporal coordinates (2.1.5)
Regular segment	Portion of the trajectory delimited by a pair of (x,y,z,t) coordinates called cusps and approximated by a straight line (2.1.5)
Resynchronization	Recomputation of the estimated aircraft trajectory when the trajectory is inconsistent with the aircraft's recent radar track history (2.1.1)

Root (of a tree)	The (unique) node with level zero (and no parent) (2.1.8)
Sector	Third level division (see "Center," "Area") of the Continental United States airspace to which a controller is assigned; a region bounded horizontally by a polygon and stretching vertically from a floor (the ground or a specified altitude) to a ceiling altitude (1.4.1)
Segment	A regular segment or maneuver envelope segment (2.1.4)
Segment chain	Sequence of segments modelling a trajectory through the planning region (2.1.4)
Sparse grid chain	The grid chain consisting of cells selected as occupied using the sparse criterion (2.1.7)
Sparse Subject Tree	A tree generated from an subject's sparse grid chain; it is used for subsequent maintenance operations on the Allobject Tree (2.1.8)
Subblock (of a block)	Same as "octant" (2.1.8)
Subfunction	Second level unit in the breakdown of AERA (see "Function," "Component," "Element") (1.3)
Subject	The aircraft whose new, updated, revised, or alternative (trial probe) trajectory is currently being tested by FPCP (1.5.2)
SWP	Sector Workload Probe (2.2.7)
TCAS	Traffic Alert and Collision Avoidance System (1.4.1.2.3)
Time horizon	Time bound on FPCP consideration of future trajectory information (2.1.2)
Trajectory	Description of an aircraft's position in (x,y,z,t) space, produced by applying altitude and timing assumptions to the filed flight plan and revised when necessary (1.4.1.1)

Trajectory update	One of three events: 1) a trajectory is added, 2) a trajectory is resynchronized, or 3) a trajectory is amended (2.1.3)
Tree	A graph (set of (tree) nodes connected by edges) with certain properties. Each node is assigned a level (integer); each edge connects nodes whose levels differ by 1; a single node has level 0; no node has edges to more than one lower-level node (2.1.8)
Tree node	(also called node) An endpoint of an edge in a tree (2.1.8)
Trial probe	A test using FPCP on a flight plan proposed as an alternative to one which already exists (1.5.1)
Vector	Controller-directed maneuver to provide an aircraft with a change in route (2.1.5)
Vertical maneuver envelope	A set of four points (vertices) associated with a cusp that defines the vertical protection provided around an aircraft (2.2.4)
Vertex	One of the four points in (x,y,z,t) space defining a vertical maneuver envelope (2.2.4)

APPENDIX E

AERA PDL LANGUAGE REFERENCE SUMMARY

E.1 Overview of the Use of AERA PDL

The AERA Program Design Language (PDL) has been created for the single purpose of presenting algorithms in this specification document. It evolves from previous AERA uses, and from MITRE WP-81W552, "All About E," October 1981.

The description of this appendix is intended to support readers and users of AERA PDL. AERA PDL supports readable, yet structured and consistent, descriptions of algorithms.

AERA PDL Features

- Relational data tables can be defined and manipulated by constructs in the language.
- Builtin functions are used to provide routine calculations without showing all of the detail.
- Routines are used to modularize logic paths and data scope.
- Indentation is used to indicate statement grouping, statement continuation, and levels of nesting.
- Routines explicitly define data or refer to predefined data.

AERA PDL Statements

The types of statements used in AERA PDL are:

- English language statements
- assignment statements
- routine declaration statements
- data manipulation statements
- flow of control statements

E.2 Elements of AERA PDL

Keywords

Keywords are words reserved for the usage of AERA PDL. Figure E-1 presents all the keywords used in the current version of AERA PDL, grouped for convenience.

routine construction keywords

CALL	END	ROUTINE

data reference keywords

PARAMETERS	IN
REFER TO GLOBAL	OUT
REFER TO SHARED LOCAL	INOUT
DEFINED IN GLOSSARY	

data definition keywords

DEFINE CONSTANT(S)
DEFINE VARIABLE(S)
DEFINE TABLE(S)

common arithmetic builtin function keywords

AVG	MIN	ABS	EXP	COS	ARCCOS
SUM	MAX	CEIL	LOG	SIN	ARCSIN
PROD	MEDIAN	FLOOR	SQRT	TAN	ARCTAN
		SIGNUM			
		MOD			

coordinate geometry builtin function keywords

DIST	DOT	INTERSECTION
MAGNITUDE	CROSS	INTERPOLATE
DIRECTION	LINE	

set builtin function keywords

UNIQUE	COUNT	CONCAT	BOOL

FIGURE E-1
KEYWORD GROUPINGS

set operator keywords

UNION INTERSECT

table manipulation keywords

SELECT FIELDS	ALL
INSERT INTO	FROM
DELETE FROM	INTO
UPDATE IN	WHERE
	ORDERED BY
	RETURN COUNT

value constant keywords

TRUE FALSE NULL

comparison keywords

NOT	GT	EQ	ANY
OR	GE	NE	ALL
AND	LT	IS IN	
	LE	IS NOT IN	

flow of control keywords

IF ... THEN ... ELSE
CHOOSE CASE ... WHEN ... THEN ... OTHERWISE
FOR ... TO
REPEAT WHILE
REPEAT UNTIL
REPEAT FOR EACH ... RECORD
GO TO

FIGURE E-1 (Concluded)
KEYWORD GROUPINGS

Operators

The operators of AERA PDL are summarized in Figure E-2.

The Assignment Operator

- The format of the assignment statement is:
  "target" = "expression"
- The target may be any type of data allowed by AERA PDL.
- The assignment operator includes the ability to fill a table from data contained in other tables. The form of this use of the assignment operator is:
  "table_name" = "table_expression" ;

Builtin Functions

The builtin functions of AERA PDL accept either an single value or data organized into an array. The author of a routine must make it clear in comments and text what form of data is being processed by the builtin function. Builtin functions are listed in Figure E-3.

E.3 Routine Construction

The order of appearance of constructs in a routine is:

- ROUTINE -- required
- PARAMETERS -- optional
- REFER TO GLOBAL -- optional
- REFER TO SHARED LOCAL -- optional
- DEFINED IN GLOSSARY -- optional
- DEFINE CONSTANTS -- optional
- DEFINE VARIABLES -- optional
- DEFINE TABLES -- optional
- logic flow -- required, but will vary by routine.
- END -- required

Three of the constructs are noted below:

The ROUTINE Construct

- The ROUTINE construct names the routine.
- The syntax of the ROUTINE construct is:
  ROUTINE "routine_name" ;

assignment operator

A = B	A is assigned the value of B

arithmetic operators

A + B	A plus B
A - B	A minus B
A * B	A times B
A / B	A divided by B
A ** B	A to the power of B

comparison operators

A LT B	A is less than B
A LE B	A is less than or equal to B
A GT B	A is greater than B
A GE B	A is greater than or equal to B
A EQ B	A is equal to B
A NE B	A is not equal to B

logical operators

NOT A	The logical opposite of A
A OR B	Logical OR of A and B
A AND B	Logical AND of A and B

set operators

A INTERSECT B	The set intersection of A and B
A UNION B	The set union of A and B
A IS IN B	A is an element of the set B
A IS NOT IN B	A is not an element of the set B

FIGURE E-2
GROUPINGS OF AERA PDL OPERATORS

FUNCTION	MEANING
ABS(x)	Absolute value of x
ARCCOS(x,y)	Inverse cosine of the ratio of y to x
ARCSIN(x,y)	Inverse sine of the ratio of y to x
ARCTAN(x,y)	Inverse tangent of the ratio of y to x
AVG(A)	Mean of the elements in A
BOOL(x)	Numerical equivalent of logical condition: 1 if x is TRUE, 0 if x is FALSE
CEIL(x)	Smallest integer greater than or equal to x
CONCAT(s1,s2,...,sN)	Concatenation of strings s1 through sN
COS(x)	Cosine of x
COUNT(A)	Number of elements of a set A
CROSS(v1,v2)	Cross product of vectors v1 and v2
DIRECTION(p1,p2)	Direction of p2 from p1 in degrees from the north; usually will be expressed in degrees clockwise from true north
DIST(p1,p2)	Euclidean distance between points p1 and p2
DOT(v1,v2)	Dot product of vectors v1 and v2
EXP(x)	e to the x power
FLOOR(x)	Greatest integer less than or equal to x

FIGURE E-3
BUILTIN FUNCTIONS

FUNCTION	MEANING
INTERPOLATE(a,b,t)	The point (1-t)a+tb
INTERSECTION(L1,L2)	The point of intersection of the lines L1 and L2
LINE(p1,p2)	Vector (a,b,c) corresponding to the line ax + by = c which passes through the points p1 and p2
LOG(x)	Log of x in base e
MAGNITUDE(v)	Length (i.e., norm) of the vector v
MAX(A)	Largest of the elements in the set A
MEDIAN(A)	Median value of the elements in set A
MIN(A)	Smallest of the values in set A
MOD(x1,x2)	Remainder when x1 is divided by x2
PROD(A)	Product of the elements in A
SIGNUM(x)	Function yielding 1 if x GT 0, -1 if x LT 0, and 0 if x EQ 0
SIN(x)	Sine of x
SQRT(x)	Square root of x
SUM(A)	Sum of the elements in A
TAN(x)	Tangent of x
UNIQUE(A)	The set A with no duplicate elements

FIGURE E-3 (Concluded)
BUILTIN FUNCTIONS

The CALL Construct

- The CALL construct invokes use of another routine as a subroutine and passes to it the data on which it is to operate.

- The syntax of the CALL construct is:
  CALL "routine_name" ("data_usage_list") ;

- The data usage list in the CALL statement must match in number and data utilization (IN, OUT, INOUT) the PARAMETERS statement of the called routine.

The END Construct

- The END construct shows the formal end of the routine.

- The syntax of the END construct is:
  END "routine_name" ;

E.4 Data Definitions

Data usage is defined in the constructs placed at the beginning of each routine.

The structures, or organization of data, recognizable to AERA PDL include constants, atomic variables, hierarchically structured variables, arrays, tables, and field-types. The hierarchically structured variables are the same as the structure variables of PL/I.

Within a table, the values corresponding to the definition of a field-type are called fields when they are referred to individually. The values for a whole column of a table (or a subset of the whole column) may be referred to as a set of fields.

The following data definition constructs appear in the order shown, if any are needed. The first line of each construct begins in column 1, aligned with the ROUTINE construct.

The PARAMETERS Construct

- This construct provides usage information about the data that are being provided by the calling routine in the form of specification of read-only 'IN', write-only 'OUT', or modification of an existing value 'INOUT'.

- Variables appearing in the PARAMETERS construct are still local data for the routine being defined and as such appear in the definition constructs.

- The syntax of the PARAMETERS construct is:
  PARAMETERS "data_usage_list" ;

The REFER TO GLOBAL Construct

- This construct provides reference to, and usage information for, data from the Global data model.

- The syntax of the REFER TO GLOBAL construct is:
  REFER TO GLOBAL "data_usage_list" ;

The REFER TO SHARED LOCAL Construct

- This construct provides reference to, and usage information for, data from the Shared Local data model described in Appendix A of the specification.

- The syntax of the shared local construct is:
  REFER TO SHARED LOCAL "data_usage_list" ;

The DEFINED IN GLOSSARY Construct

- This construct provides reference to, and usage information for, data from a specially prepared Glossary that centralizes the definition of data variables that are used repeatedly within a given function of the algorithmic specification.

- The syntax of the shared local construct is:
  DEFINED IN GLOSSARY "data_usage_list" ;

The DEFINE CONSTANTS Construct

- The use of named constants instead of in-line numerical constants is available at the discretion of the author of an algorithm. Named constants, if present, are to be declared with this construct.

- The syntax of the DEFINE CONSTANTS construct is:
  DEFINE CONSTANTS "constant_definition_block" ;

The DEFINE VARIABLES Construct

- The syntax of the DEFINE VARIABLES construct is:

```
DEFINE VARIABLES "variable_definition_block" ;
```

The DEFINE TABLES Construct

- The syntax of the DEFINE VARIABLES construct is:

```
DEFINE TABLES "table_definition_block";
```

E.5 Flow of Control Constructs

The IF...THEN...ELSE Construct

- The syntax of the IF...THEN...ELSE construct is:

```
IF "condition"
    THEN
        "statement_block"
 [ ELSE
        "statement_block" ]
```

The CHOOSE CASE Construct

- This construct provides a choice of one of several alternative logic paths depending on the first condition satisfied among the conditions specified.

- The OTHERWISE phrase is optional.

- The syntax of the CHOOSE CASE construct is:

```
CHOOSE CASE
   WHEN "condition" THEN
        "statement_block"
 [ WHEN phrases repeated as necessary ]
 [ OTHERWISE
        "statement_block" ]
```

The REPEAT WHILE Construct

- The syntax of the REPEAT WHILE construct is:

```
REPEAT WHILE "condition" ;
    "statement_block"
```

The REPEAT UNTIL construct

- The syntax of the REPEAT UNTIL construct is:

```
REPEAT UNTIL "condition" ;
    "statement_block"
```

The REPEAT FOR EACH RECORD Construct

- This construct explicitly loops over all records in table, or the subset of a table as specified in a WHERE phrase.

- The syntax of the REPEAT FOR EACH construct is:

```
REPEAT FOR EACH "table_name" RECORD
  [ WHERE "condition" ] ;
     "statement_block"
```

- Within the statement block of this loop, the construct of "table_name"."field_name" means only the ONE value that is associated with the record for that iteration of the loop.

- If it is necessary to refer to entire columns of the table that is being looped on, the correct form of the reference is ALL("table_name"."field_name"). This construct means exactly what "table_name"."field_name" would have meant if the loop had not been in effect.

The GO TO Construct

- The syntax of the GO TO construct is:

```
GO TO "label" ;
```

The FOR...TO... Construct

- The syntax of the FOR...TO... construct is:

```
FOR "loop_index" = "initial_value" TO "last_value" ;
   "statement_block"
```

E.6 Table Manipulation Constructs

The SELECT FIELDS Construct

- This construct extracts data from a table, or from a collection of tables, and makes it available to the routine.

- The syntax of the SELECT FIELDS construct is:

```
SELECT FIELDS [ UNIQUE ] [ "field_list" | ALL]
   FROM "table_name_list"
  [ INTO "local_variable_name_list" ]
  [ WHERE "condition" ]
  [ ORDERED BY "field_name" ]
  [ RETURN COUNT ( "local_variable" ) ] ;
```

The INSERT INTO Construct

- This construct allows a new record to be inserted into a table.

- The syntax of the INSERT INTO construct is:
  INSERT INTO "table_name" ("field_assignments")
  [WHERE "condition"] ;

- All insertions will preserve the assumption of no duplicate records allowed in the table.

The UPDATE IN Construct

- This construct allows existing records in a table to have certain of their values changed.

- The syntax of the UPDATE IN construct is:
  UPDATE IN "table_name" ("field_assignments")
  [WHERE "condition"] ;

The DELETE FROM Construct

- This construct removes selected records from a table.

- The syntax of the DELETE FROM construct is:
  DELETE FROM "table_name"
  [WHERE "condition"] ;

E.7 Glossary

"comparison"

- There are four possible syntaxes for the comparison. These are not given separate names, but will all be shown as if they shared the same element of the language.

- The first syntax is for arithmetic comparisons:
  "individual" GE|LE|GT|LT "individual"

- The second syntax is for general comparisons:
  "individual" EQ|NE "individual"

- Both of these syntaxes are also valid if they are used to compare two variables with the same complex organization, for example two vectors of the same length or two field types from the same table. In this case the result has as many answers as there are elements in the compared variables.

- The third syntax is for arithmetic comparisons:
  "individual" GE|LE|GT|LT ANY|ALL "set"

- The fourth syntax is for general comparisons:
  "individual" IS IN|IS NOT IN "set"

- The latter two syntaxes are used to qualify an individual based on any value in a set of values.

"condition"

- The syntax of the condition is:
  "comparison" [AND|AND NOT|OR|OR NOT "comparison"]

- The optional part of this syntax can be repeated as often as required.

"constant definition block"

- The content of the constant definition block is three columns: the constant names, the constant values, and the constant descriptions.

- The constant names are aligned in a column 3 spaces indented from the DEFINE CONSTANTS line.

- The other two columns are aligned as convenient, so that there is no visual overlap between the columns.

"data usage list"

- A routine must declare the type of use for all of its data that are known outside the routine.

- The three types of use are: read only (IN), create (OUT), and modify an existing copy (INOUT).

- The format of a data usage list is:
  "variable_name" "usage_type", ...

- An example of the format for data usage list is:
  An_Input_Parameter IN, A_LOCAL_TABLE INOUT

"expression"

- Variables may be formed implicitly in expressions without being separately named or defined.

- Expressions are combinations of defined variables with the operators and builting functions of AERA PDL.

- In an expression, the implicit variable output from any builtin function may be used as the input to any other builtin function or operator.

- An expression, when fully evaluated, yields one variable.

"field assignments"

- This term only appears in statements referring to exactly one table: INSERT and UPDATE.

- The form of the term is a comma-separated list:
  "field_assignment", ...

- The form of a single assignment is:
  "field_name" = "value_expression"

- In this term the field_names do not have to be qualified by the table name (that is given in the statement).

"table definition block"

- Three types of definition are made in this block: table definitions, field-type definitions, and AGGREGATE definitions.

- Table definition lines are formatted as:
  "table_name" "table_definition"

- Field-type definitions lines are formatted as:
  "field_name" "field_definition"

- Aggregate definitions are formatted as:
  "aggregate_name" AGGREGATE ("field_name_list")

- Fields will contain only atomic (single-valued) variables.

- Aggregates may be used so that a program can manipulate multiple fields in one statement when it makes sense to do so.

"table-expression"

- Tables may be used implicitly in assignments or comparisons being separately named or defined.

- A table expression is either a table name or a SELECT statement specifying the contents of the implicit table.

"table name"

- Generally, this is just the name of a table.

- In a few statements, there is a syntax that allows a user to define a synonym and use it in the rest of that statement. The intent of this option is to allow shorter where clauses that are easier to read. The format of the synonym reference is:
  "existing_table_name" ("synonym")

- The statements that allow this use are those that have the where clause: SELECT, INSERT, DELETE, UPDATE, and REPEAT.

"variable definition block"

- The content of the variable definition block is two columns: variable names and variable descriptions.

- Align variable names in a column that is indented 3 spaces from the DEFINE VARIABLES line.

- Align variable definitions in a column as convenient; when a structure element is defined, both the variable name and the variable definition should be indented three spaces from the name and definition of the next higher level variable.

- Three types of variables may be defined in this block: atomic variables, arrays, and structured variables.

- Each element variable is described by a line:
  "variable_name" "variable_definition"

- Each array variable is described by a line:
  "variable_name" ("dimensions") "variable_definition"

- Each structured variable is described by multiple lines, one line per lowest level element, and one line for each named level of grouping/structure, with indentation levels used to indicate the grouping.

- The names of subordinate elements of a structured variable are named in all lower case letters.

- The use of complex structured variables is not encouraged; one reasonable use for them is to receive the values of AGGREGATEs.

E.8 Other Uses and Conventions

Use of Special Characters in AERA PDL

- Parentheses are used for grouping statements and setting off special parts of the constructs.
- Semicolons are used as statement terminators.
- Colons are used to terminate labels.
- Underscore is used to separate words in multi-word identifiers.
- The symbols '+','-','*', and '/' are used as arithmetic operators.
- The pound sign '#' is used as a comment delimiter, for beginning and end of each comment line.
- Commas are used as separators in lists of operands.
- Periods are used to separate fully qualified names.

Naming Conventions

- Keyword identifiers use only uppercase letters and are underlined. They are the only underlined identifiers in the PDL.
- Table identifiers from the relational data base also use only uppercase letters.
- AGGREGATE identifiers for combinations of fields use no uppercase letters.
- References to fields in a table, used in the normal course of reference in AERA PDL, will be fully qualified by including the table name.

Other Identifiers

- Identifiers for constants, routines, labels, arrays, and hierarchically structured variables are all be named using word-initial capitals.
- For hierarchically structured variables, all of the subordinate elements within the structure use only lowercase letters.
- For hierarchically structured variables, all references to the subordinate elements in the structure will be in fully qualified form using separate identifiers.
- Global data and shared local data can include both tables and parameters. The individual parameters are named using word-initial capitals.

Use of the Formal Constructs in AERA PDL Statements

- Statements may use formal constructs or clear English descriptions to specify the intended test or action.
- Any AERA PDL statement is terminated by a semicolon, including any English statement outside of a comment.
- Below the level of statement, some statements have a finer organization in terms of "phrases", usually occupying a line per phrase and indented one level from the first line of the original statement.

Statement Organization

- Indentation is used to indicate statement grouping, statement continuation, and levels of nesting.
- Any statement may have a label starting in column 1.
- Continuation lines are indented three spaces from the original line of the statement.
- Comments are used as needed, bracketed by the special character '#'.

APPENDIX F

REFERENCES

1. U.S. Department of Transportation, Federal Aviation Administration, Advanced Automation Systems: System Level Specification, FAA-ER-130-005B, April 1983.

2. U.S. Department of Transportation, Federal Aviation Administration, National Airspace System Plan: Facilities, Equipment, and Associated Development, April 1983.

3. William J. Swedish, Barbara C. Zimmerman, Audrey W. Lipps, J. Glenn Steinbacher, "Operational and Functional Description of AERA 1.01," MTR-83W69, The MITRE Corporation, McLean, Virginia, September 1983.

4. U.S. Department of Transportation, Federal Aviation Administration, Air Traffic Control, Order 7110.65C, January 1982.

5. William J. Swedish, "Evolution of Advanced ATC Automation Functions," WP-83W149, The MITRE Corporation, McLean, Virginia, March 1983.

6. U.S. Department of Transportation, Federal Aviation Administration, "National Airspace System Configuration Management Document: Automatic Tracking," NAS-MD-321, August 1, 1982.

7. William P. Niedringhaus and Andrew D. Zeitlin, "Collision Avoidance Algorithms for Minimum TCAS II," MTR-82W158, The MITRE Corporation, McLean, Virginia, March 1983.

8. Audrey W. Lipps, William J. Swedish, Barbara C. Zimmerman, "Operational and Functional Discriptions of the AERA Packages," MTR-83W125, The MITRE Corporation, McLean, Virginia, September 1983.

9. Gregory M. Hunter and Kenneth Steiglitz, "Operations on Images Using Quadtrees," IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol. PAMI-1, No. 2, pp. 145-153, April 1979.

10. Hanan Samet, "An Algorithm for Converting Rasters to Quadtrees," IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol. PAMI-3, No. 1, pp. 93-95, January 1981.

11. Hanan Samet, "Region Representation: Quadtrees from Boundary Codes," Communications of the Association for Computing Machinery, Vol. 23, No. 3, pp. 163-170, March 1980.

12. Charles R. Dyer et al., "Region Representation: Boundary Codes from Quadtrees," Communications of the Association for Computing Machinery, Vol. 23, No. 3, pp. 171-179, March 1980.

13. Glenn C. Kinney, Glennis L. Bell, and Martin A. Ditmore, "Glossary of Common Terms in Air Traffic Control Operations," WP-83W22, The MITRE Corporation, McLean, Virginia, March 1983.

END

FILMED

2-84

DTIC